



BOOKBYTE

Final Report

(SUPERVISOR) JAMAL

(STUDENT C00260735) CONOR

Table of Contents

Introduction	2
Project Description.....	2
Implementation of Key Functionality.....	3
Uploading PDFs for Segmentation:	3
Segmenting Reading Material:	3
Progress Tracking and Reading Streaks:.....	4
Notification system	4
Adapt Reading Segments.....	5
Algorithms Implementation:.....	5
Potential Improvements for Future Development	5
Technical Description	6
Firebase Cloud Functions	6
Android Studios	6
Kotlin	7
TypeScript.....	7
Final Screens	8
Login Screen	8
Register Screen	9
Forgot Password Screen	10
Difficulty Selection Screen	11
Custom Word Count Screen.....	12
Upload Content Screen	13
Upload Content Confirm Screen	14
Library Screen.....	15
Gutenberg Library	16
Challenges Faced	17
Firebase Cloud Function	17
Challenges With Parsing PDFs	18
Learning Outcomes	19

Introduction

This final report presents an overview of BookByte, an Android application specifically developed to improve users reading habits. BookByte addresses the challenge of managing large reading materials by implementing a core feature known as 'chunked content delivery'. This feature allows users to upload their PDFs, which the app then segments into smaller, more manageable pieces. These segments can be customized in size and complexity, evolving in response to the user's improving habit.

The idea for BookByte was to create a tool that not only encourages regular reading but also makes the activity less daunting and more adaptable to each individual reader. The app implements functionalities like reading streaks, progress tracking, and gamification to motivate users to read each day. Additionally, its notification system helps in maintaining user engagement by providing users reminders to continue their reading journey.

The overall goal of BookByte is to enhance and promote continuous learning. By simplifying the reading process and making it a daily habit, the app aims to support learning in today's digital society. This report will detail the development process, key features, user feedback, and future directions for BookByte.

Project Description

BookByte is an Android application developed to enhance reading habits among users. The primary aim of the app is to improve the development of new, consistent reading habits by breaking down text into manageable segments. This feature allows users to engage more effectively with their reading materials, growing more efficient reading practices. In this section of the report, we will detail the key functionalities implemented in BookByte to achieve this goal, highlighting how these features contribute to the app's effectiveness in promoting regular reading habits.

Implementation of Key Functionality

Uploading PDFs for Segmentation:

The PDF upload and segmentation functionality in BookByte involves three main steps:

1. **Retrieving the PDF from Android Storage:**

The app accesses the Android device storage to locate and select the desired PDF file.

2. **Conversion of PDF to URI and Transmission to Firebase:**

The selected PDF is converted to a URI and sent to Firebase for processing.

3. **Triggering the Firebase Cloud Function for Segmentation:**

The upload of the URI in Firebase Storage automatically triggers a Cloud Function that begins the segmentation of the PDF into manageable segments.

This process allows users to easily upload any PDFs to their library.

Segmenting Reading Material:

The segmentation process for reading materials in BookByte involves many steps:

1. **PDF Retrieval and Directory Creation:**

Once a PDF is uploaded, it is retrieved from cloud storage and a dedicated directory is created for storing each user's segmented PDFs. This ensures user data is separate to allow for easily accessible segments in the app.

2. **PDF Validation and Segmentation:**

The PDF is checked for validity. After confirmation, the cloud function retrieves the user's word count from Realtime-DB, and the PDF is segmented. This process adapts the reading segments to the user's preferred word amount.

3. **Adaptive Segmentation with Segment Adapter Algorithm:**

If the segment adapter algorithm is triggered by updated user preferences or, dynamically within the Android app, the segmentation process is also triggered creating new segments with updated sizes. These replace the previous segments to adapt to the user's improving reading habits.

Progress Tracking and Reading Streaks:

The progress tracking and reading streak features in BookByte are designed to motivate users by monitoring the progress of their reading habits:

1. **Updating Reading Streaks:**

A reading streak is updated each time a user completes their daily reading segment. The streak can only be updated once per day, and the app tracks this timing to ensure accuracy.

2. **Time Tracking for Reading Segments:**

The app records the time it takes for a user to complete each reading segment. These times are then stored in cloud storage for future analysis and reference.

3. **Data Utilization and Analysis:**

The recorded times are used to generate a graph displaying the user's reading times, highlighting their fastest and slowest times. This visual graph helps users understand their reading patterns and encourages improvement.

4. **Tracking Overall Progress:**

Additionally, the app keeps track of the total number of segments a user has completed and records the longest reading streak achieved.

Notification system

1. **Setting Notifications with AlarmManager:**

The app utilizes Android's AlarmManager to schedule notifications at a specific time each day. This ensures that users receive a consistent reminder to engage with their reading material.

2. **Notification Channel Creation:**

A notification channel is created to manage how notifications are shown to the user. This channel helps in organizing notifications.

3. **Reading Streak Reminders:**

The primary purpose of these notifications is to remind users to maintain their reading streak. By encouraging daily interaction, the app aims to foster consistent reading habits.

Adapt Reading Segments

This feature is a critical component of the application, dynamically adjusting the segment size in response to users reading metrics. Below the implemented algorithm is discussed along with potential improvements.

Algorithms Implementation:

This algorithm can be potentially triggered after the completion of a daily reading segment. The algorithm gathers metrics accessible within the app and from Firebase storage to calculate the user's reading difficulty level.

Metrics Passed Into **generateSegmentSize()**:

- **currentStreak**: Reflects the user's consistency. A higher streak indicates readiness for increased segment size, as it indicates regular engagement this is controlled by different ranges and also different milestones e.g., reaching 7 days gives small segment boost. The streak increase eventually caps out at a maximum value.
- **daysMissed**: Reflects lapses in reading habit. More days missed signals a need for easier or shorter segments to rebuild the user's habit.
- **timeToCompleteSegment**: Directly measures how long it takes a user to finish a segment. Longer times suggest that the segment is too challenging. Also compared to national average and user set reading averages.
- **averageCompletionTime**: Provides a baseline for comparison against individual segment completion times, helping to adjust future segments to better fit the user's pace.
- **onSameDay**: Tracks whether the segment was completed on the same day another segment was completed. Reading many segments a day indicates the current segment size is too manageable.

The algorithm decides if an increase or decrease in difficulty is needed after calculations. The app prompts users that segments have been recalibrated.

Potential Improvements for Future Development

Use Of Machine Learning:

Implement a machine learning model to predict optimal segment sizes based on user engagement over longer periods, utilizing changing reading abilities and schedules.

Further User-Tested Segment Adjustments:

Further user testing to gather more data on how different segment sizes affect reading consistency and engagement. Utilize this data to fine-tune the algorithm, ensuring it better aligns with different user preferences

Technical Description

Firebase

In the development of BookByte, multiple features of Firebase were used to improve functionality and user experience. Firebase Authentication was added to manage user sign-ins and sign-ups securely, ensuring a safe environment for users to store and access their data. Firebase Storage was crucial for handling large files, particularly for the uploading and retrieval of user PDFs, providing file management within the app. Additionally, Firebase Realtime Database was integrated to provide real-time data synchronization across all user devices. This feature allowed for instant updates of reading segments, user progress, and other dynamic content, so that the user experience was very responsive. The combination of these Firebase services provided a robust backend infrastructure that supported the requirements of BookByte, from user management to data storage and real-time updates.

Firebase Cloud Functions

Firebase Cloud Functions has a massive role in the execution of most key functionality in BookByte. It provides a serverless computing solution that executes backend code in response to events triggered within or outside of Firebase. This technology was used for its flexibility and scalability, which are essential for handling tasks like data processing and dynamic segmentation of PDFs. By using Firebase Cloud Functions, BookByte doesn't have to perform heavy data processing on its users devices which allows that app to feel more responsive and improves the apps overall performance. This serverless approach not only reduces overhead costs but also enhances the app's ability to scale according to user demand, ensuring a responsive service for generating users reading segments. Cloud functions also provided an easy way to modify data in services such as Firebase Storage or Realtime Database

Android Studios

For the development of BookByte, Android Studio was used as the primary integrated development environment (IDE). Android Studio's is the easily the best, and maybe only, choice for Android application development. It offers a large platform for designing, testing, and debugging, which helped facilitate the development of BookByte's intuitive user interface and the easy integration of its functionalities. Android Studio's support for native Android

technologies ensured that BookByte could use the full capabilities of the Android operating system, resulting in a more efficient application. The IDE's built-in features, such as the emulator, notification system, UI/UX design tools and code analysis tools, allowed for a smooth development process. Although I had limited experience with this software before starting the development of my app, I had no difficulties diving right into the development process.

Kotlin

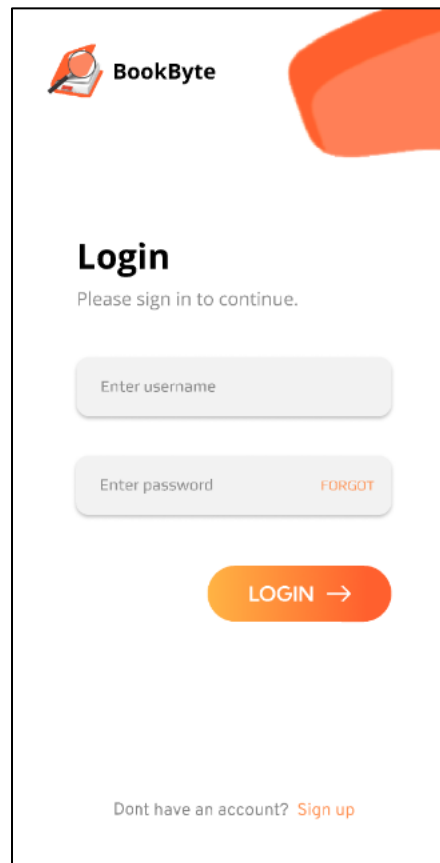
For the development of BookByte, Kotlin was selected as the programming language, despite being new to me. The decision to use Kotlin was due to its easy integration with Android Studio. Kotlin's concise syntax and new features, such as the removal of getters and setters, made it an improvement over Java. The language's ability to reduce boilerplate code significantly helped in the creation of a robust and reliable app. Kotlin's safety features, like nullability checks, also contributed to creating a more stable app, making it a better overall choice for developing BookByte.

TypeScript

TypeScript was chosen to program the functionality of the Firebase Cloud Functions used in BookByte. The use of TypeScript allowed for more maintainable and error-resistant code, making it ideal for handling the complex logic required by Firebase Cloud Functions. This programming language allowed for the development of efficient functions that respond to real-time events within BookByte.

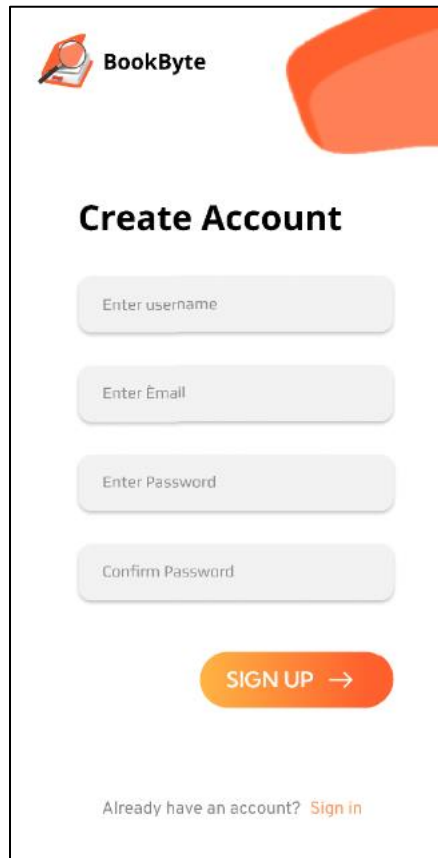
Final Screens

Login Screen



The login screen is the initial activity loaded when BookByte launches. Here, users are prompted to enter their username and password to log in. For new users wanting to join the app, a 'Sign Up' link is conveniently positioned at the bottom of the page, which redirects them to the registration screen. Additionally, the screen includes a 'Forgot Password' feature, providing a straightforward way for users to recover access to their accounts should they forget their credentials. This setup ensures a user-friendly and accessible interface, facilitating smooth navigation for all types of users.

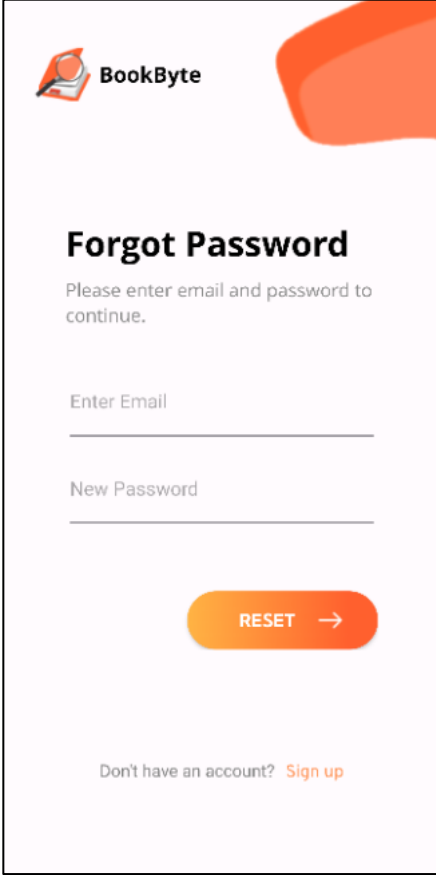
Register Screen



The image shows a mobile application register screen for 'BookByte'. At the top left is the BookByte logo, which consists of a stylized orange and white icon of a book and a magnifying glass, followed by the text 'BookByte'. The background features a large, abstract orange shape in the top right corner. The main heading is 'Create Account' in bold black text. Below this are four input fields: 'Enter username', 'Enter Email', 'Enter Password', and 'Confirm Password', each in a light gray rounded rectangle. A prominent orange rounded button with the text 'SIGN UP →' is positioned below the input fields. At the bottom, there is a link: 'Already have an account? [Sign in](#)'.

The register screen is designed for new users to create their account on BookByte. Here, users are required to input their details to sign up, including their username, email address, and a password. This screen maintains the same color scheme and layout as the login page, providing a consistent user experience. However, it requires additional information to ensure each account is uniquely identified and secured. This consistency in design helps users navigate the sign-up process smoothly while distinguishing between login and registration functionalities.

Forgot Password Screen



BookByte

Forgot Password

Please enter email and password to continue.

Enter Email

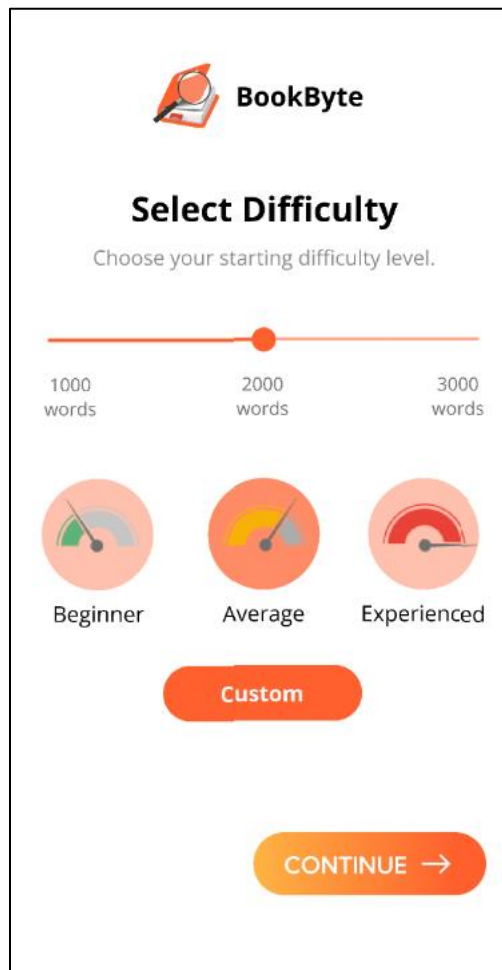
New Password

RESET →

Don't have an account? [Sign up](#)

The Forgot Password screen offers a straightforward way for users who need to regain access to their BookByte account after forgetting their password. Upon entering this screen, users are prompted to input the email address associated with their account. After the email address is submitted, Firebase Authentication sends a password reset link to that email inbox. Users can then follow this link to securely change their account password. This process is designed to be user-friendly and efficient, ensuring that users can quickly restore access without unnecessary complications.

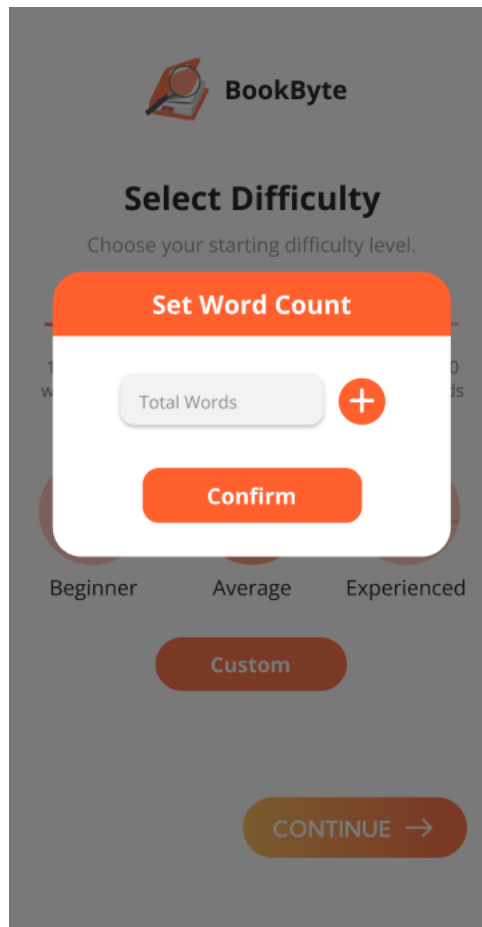
Difficulty Selection Screen



The difficulty level screen allows users to choose between four different levels of difficulty: beginner, average, experienced and custom. After entering this screen, users are prompted with the four difficulty levels, each designed to tailor the user's skill level and preference.

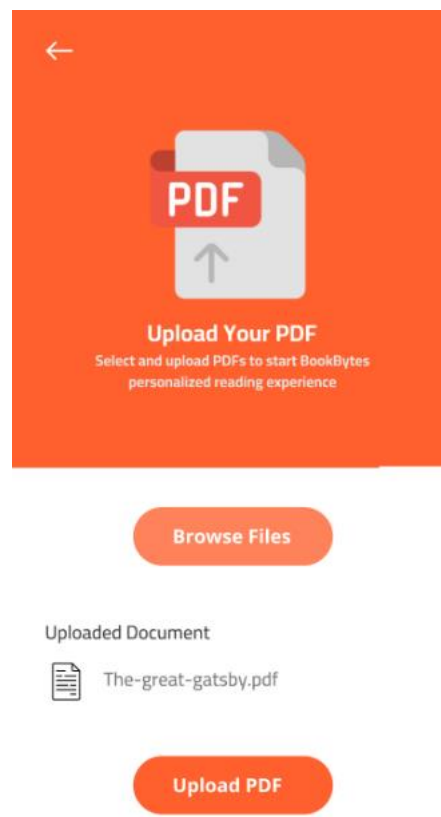
Users can select their skill level by entering the number of words they want to read daily manually in the custom section or by using the slider. The slider has predefined word metrics ranging from beginner to experienced.

Custom Word Count Screen



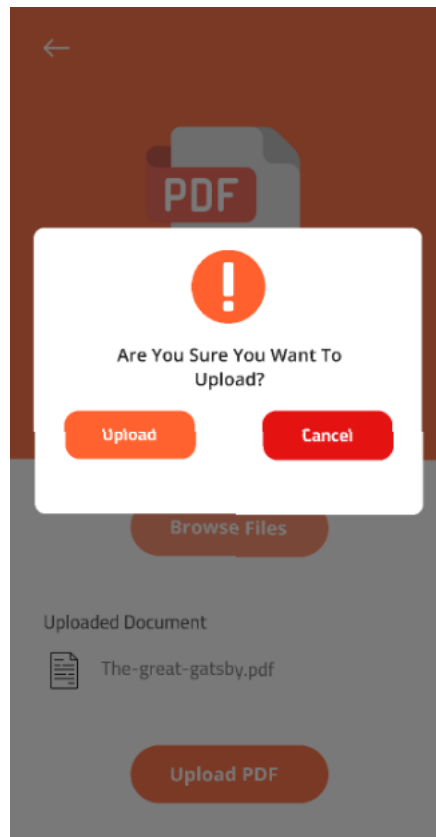
After a user clicks the custom button on the difficulty selection screen a smaller screen opens, giving the option for the users to customize their word count goal. After opening this screen users find a box where they can manually input their preferred word count. Adjacent to the input box there is a plus button, which users can use to adjust the word count, increasing it by 10 each time the user taps it. Users can also hold down the button to reach their desired word count. After the user has entered their desired word count, they can then save their selection by clicking the 'confirm' button located at the bottom of the screen.

Upload Content Screen



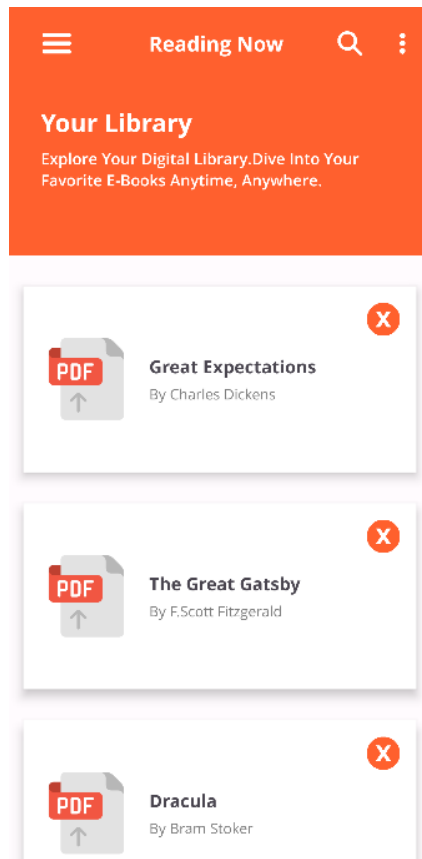
The upload content screen enables users to upload reading materials to the app's library. After opening this screen users are presented with the option to upload a PDF, such as a book, article, or document. Users can upload a PDF by tapping on the 'upload PDF' orange button located in the bottom half of the page.

Upload Content Confirm Screen



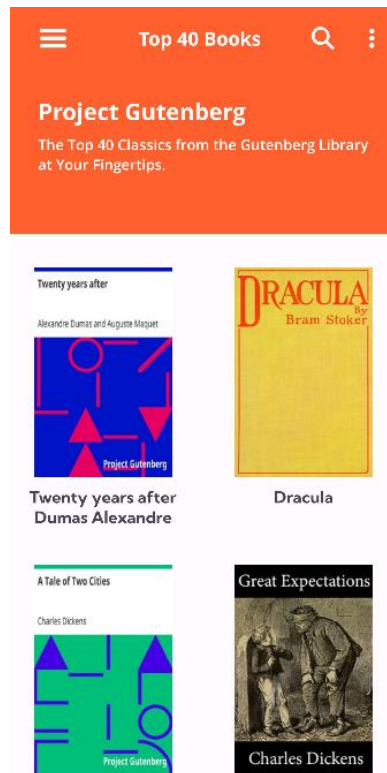
After uploading a document, the user is brought to a smaller screen within the upload content screen, asking them to confirm their upload decision. The screen displays a simple prompt asking the user “Are you sure you want to upload?”, users are then given two options: “Upload” to confirm the upload or “Cancel” to cancel the upload.

Library Screen



The library screen acts as a central hub where users can access their collection of reading materials within the BookByte app. After entering this screen users are given a visually organized display of all their uploaded documents, sorted alphabetically by title. Each document displays a title and an author and also includes a close button indicating the deletion of a book. Users can browse through their library in alphabetical order or by using the search bar at the top of the page.

Gutenberg Library



The Gutenberg library within the BookByte app provides users with a collection of free eBooks from Project Gutenberg, it is a well-known digital library offering over 60,000 public domain titles.

Challenges Faced

Firestore Cloud Function

Throughout the development of BookByte, integrating Firestore Cloud Function features presented a lot of challenges:

Learning Curve:

The initial challenge was researching how to utilize Cloud Functions for the segmentation functionality I was trying to achieve. This involved understanding the setup, deployment, and management of Cloud Functions. I had to learn how Cloud Function could work for my specific use case and how I would test this to find out.

Small Storage Connection Issues:

Early in development, there were challenges getting a connection between Cloud Functions and Firestore Storage. Due to an incorrect bucket URL in admin configurations in Cloud Function causing early frustration as I couldn't send or receive any data from Storage.

Minor Dependency Issue:

A dependency issue in Android Studio related to Firestore prevented early testing of segmentation functionality in Cloud Functions.

Debugging Delays:

Debugging the Cloud Function code was particularly challenging due to the slow deployment process. Each change required redeployment, significantly slowing down troubleshooting and testing.

Familiarization with Google Console Logs:

Adapting to the Google Console Logs interface was necessary to monitor and debug functions, but this too required a learning period to understand and begin to use effectively.

Project Gutenberg Challenges:

Challenges around retrieving data from Project Gutenberg and splitting it into images, metadata and pdfs caused a lot of wasted time constantly fixing and redeploying to get it working. Eventually had to use a separate API Gutendex.

Segmentation Code Issues:

Numerous problems with the segmentation code, designed to divide text into manageable reading segments. Getting this functioned correctly was critical for the core reading experience offered by the app and was one of the biggest challenges faced in this project.

Challenges With Parsing PDFs

Parsing PDFs to segment text by size was the greatest technical challenge during the development of BookByte. The core issue was extracting text from PDFs while preserving original formatting elements such as headings, fonts, bullet points, and images. This preservation was crucial to display the parsed text accurately within the reading segment viewer.

Initial Attempt with PDF.js:

The initial strategy involved using the PDF.js library to extract the PDF's format and save it as a JSON file. I then attempted to re-parse the extracted text using a JSON object. While this approach was partially successful, it was highly inefficient—segmenting text for each reading segment took nearly five minutes, rendering the process impractical for real-time use. Plus having to wait for the segment, change code in Cloud Function, redeploy code, and then segment text again, wasting nearly 10 minutes every time I wanted to try modifying any code.

Switch to Apache PDFBox:

In search of a more efficient solution, I tried the Apache PDFBox library, which allowed for saving text formatting in an XML layout. However, this method required setting up a Docker container and hosting it on Google Cloud, which introduced additional complexity and maintenance challenges.

Challenges with PDFBox:

Despite the potential of Apache PDFBox, I had significant difficulties in getting the PDFBox container to function correctly. Also, trying to work between three different environments was frustrating and confusing. The time and resources spent trying to get this solution working ended up becoming wasteful, leading me to abandon this approach before I left it too late.

Final Solution with pdf-lib:

Eventually, I implemented the pdf-lib JavaScript library, which provided a more straightforward solution to the segmentation problem. This library allowed for effective text segmentation without the need to preserve complex formatting, aligning better with the project's needs.

Learning Outcomes

Learn How to Develop an App:

I got experience working on all aspects of developing a functional Android app and Firebase Cloud Functions. This provided me with hands-on experience in handling all stages of app development, from coming up with the idea to completion.

Chance To Try Agile Practices:

This project allowed me to learn how to develop an app in an agile way. I participated in regular meetings with a supervisor, followed strict deadlines for app documentation, which I got a chance to improve upon over each iteration. This iterative process of development allowed me to grasp a better understanding of the agile.

Learned New Technical Skills:

I learned to use Android Studios and Kotlin, a new programming language for me. I got a chance to try many different libraries and technologies throughout the development of the app e.g., Docker. Additionally, I developed my skills in designing user interfaces using XML layouts, which improved my technical and design skills massively.

Firebase Features:

The project was an opportunity to dive into the many great features of Firebase, gaining valuable skills in a widely used backend service. This experience could be particularly beneficial for my future projects and possibly my development career.

Documentation Practice:

I got to create and improve upon several types of documentation common in software development, such as functional specifications, design documents, and various reports. Setting both functional and non-functional requirements helped me in understanding and defining what was needed for the app to succeed.

UI/UX Design:

Improving my UI/UX design skills was a crucial part of this project. I learned to better understand user needs and ensure the interface was both appealing and functional, focusing on creating an easy to use and engaging user experience.