University of Birmingham

MSci Computer Science project

Building Processing Architecture And Designing Visualisation For Literature Works

Roman Podkovyrin

Word Count: 12829

Supervised by
Prof. Ian Batten
School of Computer Science

August 9, 2022

Acknowledgments

I would like to express my deepest appreciation to all those who provided me with the possibility to complete this project. I give special gratitude to my final year project supervisor, Prof. Ian Batten, whose contribution in stimulating suggestions and guidance helped me coordinate my project.

Special thanks go to Duncan Hui and Elliott Wright, who participated in informal interviews as authors and have kindly allowed me to borrow their books for this project.

Furthermore, I would also like to acknowledge the help of Alex Nastase for acting the role of the reader and giving back valuable feedback.

Contents

List of Figures

List of Tables

Abstract

This paper will present an architecture design for book visualisation using Natural Language Processing (NLP) techniques for extracting Named-Entities of both characters and locations using the CoreNLP library. We will also examine how this information will be used to visualise character appearance and interaction with other characters in the book. We will discuss its usefulness as an alternative summary of the book that can be consumed visually and allow us to extrapolate the book's story, focusing on both readers and authors as the target audience. Processing server architecture was used to offload heavy computations from mobile client applications, making this tool usable for the general public who don't have the strong technical knowledge or powerful machinery to run complex algorithms.

The final system has achieved all set-out goals regarding its architecture and visualisation targets. Although the target audience didn't find it extremely useful as a tool, they all agree that the system was appropriately designed and offers insight into books. There is tremendous potential with additional functionality, which can be built on top of the architecture presented here.

Several alternative approaches were introduced to be explored in future projects, namely entity clustering to improve interaction estimation between characters and a visualisation proposal that uses lessons learned and combines character and locations into one visualisation.

1 Introduction

When it comes to literature books, they can range from short stories to complex multi-book epics. Despite all the genre differences, there is one constant in all of them. They all contain at least some form of character that we follow. Some books might only mention a few, whereas others (like "The Great Gatsby - F. Scott Fitzgerald") might mention many characters. Those characters will go through several changes, be involved in storylines and interact with other characters throughout the book. Some readers and authors might experience difficulty remembering all the character names and storylines.

The process of reading or writing books is usually done over many sessions. Some of the information will need to be recalled when resuming the session, such as the characters from the last chapter, the location or some plot points. One way of recalling this could be to read the last few paragraphs of the last chapter. However, this is time-consuming and doesn't always help remember the whole chapter or book.

An alternative would be to make handwritten notes and summaries to help remember. Such as character profiles, timelines or mindmaps. This, however, is burdened with a number of drawbacks. Such as a need to carry a notepad with the book everywhere, to make notes. It will be exerting mental energy to draw and organise those notes. It might not be reasonable to make notes all the time, especially if the session is only 10-15 minutes long.

Several resources can be found online that contain chapter by chapter summaries; however, the authors of those are not always trustworthy or correct. Additionally, if reading a less popular book or writing your own, there won't be any summaries you can find online.

What if we could create a tool that automatically produces those summaries from the book itself? Could those summaries be presented visually? Additionally, how would this tool need to be designed architecturally to allow anyone to use it? This project will look into ways and methods of building such a tool that could be useful for both authors and readers when working on or reading a book.

This report covers much ground. Therefore some interesting sections are highlighted here:

- Section 1.4: Plan and Deliverable - Walks through what the project is setting out to do and defines requirements.
- Section 3.1: Android App - Describes Android Architecture.
- Section 3.2.5: Processing - Testing Processing Pipelines to find balance between quality and performance.
- Section 4: Security Incident - Talks about a security incident, how it worked and how it was patched.
- Section 6: Literature Visualisation breakdown - Talks through some of the decisions made when making visualisation. Includes recommendations for future work 6.3.4.

1.1 Related Works

The previous section has identified the problem this project is tackling. We need to be careful when doing research in this space; As there are already several papers that look into Natural Language and Literature Processing: There has been some research into text summarisation using machine learning (Neto, Freitas, and Kaestner, 2002) , Character Interaction Visualisation in a TV episode as a timeline chart (Tapaswi, Bauml, and Stiefelhagen, 2014), Literature Named Entity Recognition algorithms (Brooke, Hammond, and Baldwin, 2016) and several Information Retrieval Tasks, such as character relationship, role and sentimental analysis of characters (Valls-Vargas, Ontañón, and Zhu, 2013, Uday et al., 2020 Jacobs, 2019, Fernandez, Peterson, and Ulmer, 2015)

However, most of those papers only discuss algorithms and techniques, rather than making a usable tool for the general public, without needing any technical knowledge or powerful machinery. A big drawback of most research papers from above is a lack of reproducibility, as the code associated with the paper can't be easily run. One exception found was the Gutentag tool (Brooke, Hammond, and Hirst, 2015), which in simple words, is a corpus reader with build-in tagging. This highlights the lack of tools that allow non-programmers to use NLP techniques either for their research or while reading or writing a book. A notable example of a very popular application built on top of NLP techniques is Grammarly, which helps improve grammar and writing style. Nevertheless, even with a tool like Grammarly, performance issues have been noted during the write-up of this report. It would frequently freeze up the browser due to the document size when working on a laptop.

Popular discussion forums have been used to see if readers have problems while reading books, as there were no academic papers that covered it. Subreddit r/Books, with 20 million members, is dedicated to discussions about anything related to books. From this forum, it was found that readers have difficulties remembering events within the book even if they have read it recently (*Not long after reading a book, I forget majority of what the book is about! Is this common?*, 2014). Readers also highlight issues of remembering the characters' names, especially when there are many of them (*Struggling to understand stories and remember characters*, 2018). Most of the suggestions to help those issues mention making summaries, notes and mind maps. Therefore, readers experience difficulties while reading, and a tool that produces some form of summaries would be a great help.

When looking into the usefulness of literature visualisations, according to (Nishihara, Ma, and Yamanishi, 2021), there is some benefit to information recall. Their visualisation displayed a timeline with character appearance and their location. In the experiment, three groups were given a quiz about a novel they had read seven days prior. Only group C was allowed to look at the visualisation of the novel before the quiz. As a result, this group outperformed all the other groups in the quiz. Moreover, even doing better than those allowed to quickly skim the novel before the quiz. Visualisation has improved their answers by 28%, giving strong evidence to suggest that visualisation is a helpful form of an alternative summary.

Therefore given the currently existing projects and literature in this area, there is a gap in making literature analysis and visualisation tools that utilise NLP techniques and don't require any technical knowledge or powerful machinery for complex computations. There is also proof that some readers struggle with remembering events and characters while reading, as well as, evidence that suggests that literature visualisation can be helpful as an alternative summary.

1.2 Project Aims

The project has two distinct aims. The first is to build an architecture for book processing using NLP techniques while reducing user entry requirements regarding technical knowledge and machine specifications. The second is to look into building book visualisations with character appearances on a mobile application.

1.3 Target Audience

Two groups were identified as target audiences: Readers and Authors.

As mentioned in Section 1.1, the common issue highlighted by readers was remembering events and characters. An informal interview was conducted with Alex Nastase as a reader to confirm this hypothesis and get some further insight. He has expressed his difficulty with remembering character names. According to him, this happens because characters only appear on paper, and it's hard to associate them. Additionally, nicknames, alternative names and titles add extra confusion.

When it came to authors, it was interesting to see if they faced similar issues while writing their books. Two authors were interviewed (Duncan Hui and Elliott Wright) who are writing their first books. In a number of the informal interviews, we have discussed some of the struggles they have while writing a book and how they are trying to mitigate them. One of their main struggles is remembering the story's overall structure, the characters' personalities, and what happened to them. Writing books is not very straightforward, and authors are always rewriting, removing or adding new text, it can become tough to remember the current story. They could remind themselves by reading the whole book, but it's impractical and time-consuming. Therefore they make handmade character profiles, mind maps and chronological timelines to remind themselves. Those are the same techniques that some forum users have recommended to readers struggling while reading.

Those earlier interviews allowed to gain a much better understanding of the requirements for the system. Therefore, both authors and readers face similar problems remembering characters and events in the book. As well as both using similar techniques of helping them remember by using handwritten notes, mindmaps, etc. When describing the project, both groups have shown interest in having a tool for generating visualisations that helps automate some of the processes of getting those handwritten summaries. Although the final product was only partially useful, they found the system appropriately designed. They want more

advanced functionality that can easily be built on top of the architecture presented in this report.

1.4 Plan And Deliverable

This section will introduce the initial plan for the project, how it changed, and the final system description.

The original plan for this project was to make an Android application that highlights characters and locations with my implementation of the Named-Entity Recognition algorithm. The NER system was planned to be built with the Hidden Markov Model POS tagger or Brown Clustering (Brooke, Hammond, and Baldwin, 2016). However, building a NER implementation from scratch that gives comparable results to those implementations in other research papers would stretch beyond the time horizon for this project. Therefore it was decided to go with premade NLP libraries with NER functionality.

Two libraries were selected for testing: Apache OpenNLP and Stanford CoreNLP. Although (Brooke, Hammond, and Baldwin, 2016) found that CoreNLP produced better tagging results than OpenNLP when it came to literary works. It is worth testing both of them to see if a simple API could be used that might still produce satisfactory results while minimising performance overhead, as the tool was planned to be built for mobile use. LitNER (Brooke, Hammond, and Baldwin, 2016) was also considered, but there is no usable API, and it hasn't been maintained in a while.

The first was Apache OpenNLP, a fairly lightweight NLP tool that could easily run on any Android device locally. However, it produced unsatisfactory results, always missing obvious character and location names in unambiguous paragraphs. Therefore this library wasn't fit to analyse complex literary works.

The second library was Stanford CoreNLP, which has a larger packet size (about 507.2 MB) and is a lot more computationally expensive to run in terms of RAM and CPU usage. Unfortunately, it is not suitable to be run on mobile devices due to its size and computational requirements. The package size requires the user to have a phone with enough space to store it, and 488.2 MB (Figure 1) is quite a large app size compared with other apps as the Google Play store enforces a compressed download size restriction of 150 MB for apps. The size can be reduced by removing unused modules from the CoreNLP, but there is only so much that can be removed. An additional disadvantage of a larger app size is the increase in compile time for the project during development.

The RAM usage for CoreNLP might range from 1GB to 4GB, depending on the settings. Although there are phones with more than enough RAM on the phone to run it on, some of which can be up to 12GB, this goes directly against one of the project's goals of relaxing user entry requirements and not making users buy expensive hardware.

Overall, OpenNLP is a lightweight library that is easy to run but produces poor results

in tagging named entities. CoreNLP, on the other hand, works well for literature but is too huge and computationally expensive to be run on a mobile device. Therefore with this information in mind, it was decided to use the server for book processing, which would offload the computational load of the user's phone and make the app more lightweight by running the CoreNLP package on the server.

Originally there were no plans for producing any types of visualisation. However, after a literature review and talking to authors about what they would be interested in, visualisation became the project's main focus.

APK contents	Size
No NLP Libraries (Base)	6.4 MB
CoreNLP	488.2 MB
OpenNLP + (models for NER)	7.9 MB

Table 1: APK sizes of the app with different NLP libraries

1.4.1 Functional Requirements

From the information acquired above, we can break it into specific requirements and represent them with user stories.

There are two overlapping user roles for this application:

Reader. A person reading a book. They don't know what will happen in the story and possibly don't remember all of the characters within the book.

Author. A person who has written or is in the process of writing a book. They have extensive knowledge of the story and characters, but from constantly modifying the book, they don't remember what version of the story is currently written.

Although both have different requirements, most of the requirements will overlap. To help prioritise, requirements were labelled with the MoSCoW method (*MoSCoW method*, 2022).

Reader

As a reader, I want to ...

- RFR1. (**must**) ... load any EPUB book into the app.
- RFR2. (**must**) ... delete a particular book from the app.
- RFR3. (**must**) ... view the loaded book and change pages.
- RFR4. (**must**) ... see characters and locations highlighted in the text.
- RFR5. (**must**) ... click on the entity and see in which chapters the entity appeared.

- RFR6. (**must**) ... see visualisations about entities in the book.
- RFR7. (**should**) ... see a list of all the characters and locations within the book.
- RFR8. (**could**) ... close the book and have the app remember what page I've left off.
- RFR9. (**could**) ... see all information about characters and locations only up to a point in the book that I am now.

Author

All of the reader requirements also apply to the author.

As an author, I want to ...

- AFR1. (**could**) ... compare two different versions of the same book by using their visualisation.
- AFR2. (**won't**) ... see sentimental analysis of each character.
- AFR3. (**won't**) ... see all the actions a character has performed.

The **won't** requirements have been mentioned here to highlight that this was something authors were interested in, but this is out of the project's scope. Therefore those won't be attempted.

1.4.2 Non-Functional Requirements

Non-functional requirements have been identified that affect implementation choices.

- **NFR1 Performance**. Book processing and overall performance shouldn't take long (More than an hour). If there is no way to avoid waiting times, the implementation should simplify it for the user to wait without exerting extra attention to the app.
- **NFR2 Reliability**. In case processing takes too long due to the size of the book, the process should be killed, and the user notified.
- **NFR3 Security**. Books should only be stored locally on the device and shouldn't be stored on the server. This is to prevent books from being stolen and not break the terms and conditions of books. Appropriate security measures should be implemented to protect book information during communication with the server, such as SSL.
- **NFR4 Localization**. The system should only be functional for the English language and doesn't have support for other languages in either the UI or the book processing.
- **NFR5 Learnability**. The interface should be simple for people to understand and use.
- **NFR6 Efficiency**. It should only take a few clicks (decisions) for the user to reach their desired state.

- **NFR7 Technical knowledge of NLP**. As identified in the previous sections, there should be no programming or technical knowledge required in setting it up. Such as installing libraries, setting up environments, etc.
- **NFR8 Hardware requirements**. The application should not require the user to have a phone with high specifications to use the app, such as large RAM, processing power and storage.

1.4.3 Final Deliverable

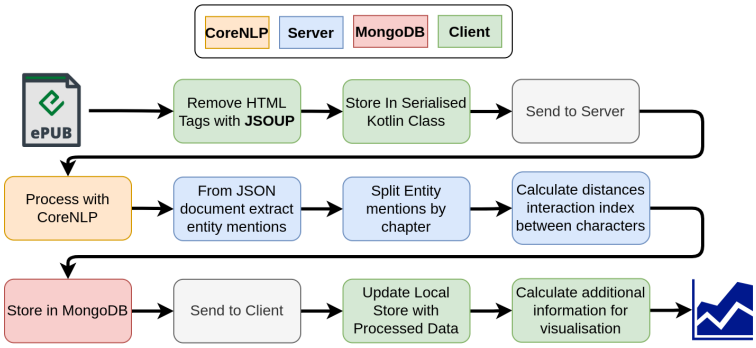


Figure 1: The diagram shows a sequence of book processing from EPUB file to phone visualisation.

The end system has multiple components that enable book visualisation: Android App, Processing Server and Development Pipeline (Figure 1).

The Android app is designed to be as light as possible regarding its size and computation requirements. All it does is preprocesses the EPUB books locally before sending them off to the server for NLP processing. Once the book has been processed on the server, the app retrieves it and performs additional computations to produce book visualisations on the phone. The app UI is fairly simple, closely mimicking most reader apps on the market. It offers a simple management page for adding, removing and opening books. The reader panel allows to switch between chapters, scroll the book and has additional elements for getting book visualisation and insights. Read more about the UI in the following Section 5.

The Server processes books with the CoreNLP NER tagger, which finds all character and location entity mentions. Additional post-processing is done to extract entities' locations in the text and calculate interaction weighting between characters. The server also stores processed books in a NoSQL database, so they can be retrieved in the future without needing to process them again.

A CI pipeline has also been produced, which simplifies the development process and makes it easy to deploy servers anywhere with minimal setup. This has been achieved with GitHub Actions and by dockerising the server modules.

2 WorkFlow and Dataset

This section discusses methods and techniques used during development and planning. Additionally, it covers the choice of the dataset for the project.

2.1 Workflow

When working on a project of this scale, there needs to be some formal plan or a workflow that you follow to ensure you deliver.

2.1.1 Agile

Throughout this project, the agile methodology was followed and *Jira* used for keeping track of tasks (Appendix A). It was very beneficial to split tickets into more manageable portions. It simplifies getting started on the task and gives a sense of progression. This process also helped avoid scope creep as the aim was to produce the minimal viable product rather than a fully-fledged solution. As a result, after six months of development and working on various technologies and concepts, a system was produced that achieved all the major functional and non-functional requirements.

2.1.2 Git workflow

Using just the master branch for a large project is not enough. A feature branch workflow was used to ensure the project had a smooth delivery. Instead of directly pushing to the master branch, a feature branch would be created that would correspond to a ticket in *Jira* (**feature/fypu-<jira-number>-<description>**).

Each pull request would automatically get prefilled with pull request template text (can be found in the repository under `.github/pull_request_template.md`), encouraging to test, comment, and document all the changes to the codebase. Pull requests also allowed to review the code before committing it to master. When merging it into master, **Squash and Merge** was used to keep commit history clean, making it easy to go back in history if something got broken.

To make commit messages clear and descriptive, they would be tagged in the following format **<commit-tag>: <Commit message>** (Appendix B).

This workflow ensured multiple opportunities to catch mistakes before committing to the master branch and allowed to work on multiple features simultaneously. As well as giving a straightforward way to undo changes if something got broken.

2.2 Dataset

As a dataset, the EPUB books were needed to test the tool's implementation and visualisation. Unfortunately, buying modern digital books is somewhat limited due to Adobe DRM protection, which prevents accessing the original EPUB file. Even if managing to get rid of the encryption with Adobe DRM API, there will still be limitations with storing and processing modern books due to terms and conditions.

A much easier and legal way to get an EPUB dataset is to use www.gutenberg.org for Royalty-free books. The pros of using this library: you get access to EPUB books without any limitations for processing, modifying or storing. However, the constraint is that there are no recent books and only classics, which limits the NLP processing due to a very different and outdated language. Books from this service also have inconsistent formatting (having multiple chapters per page, or a chapter is broken down into various parts). The Gutenberg project books also have text at the beginning and end of the book with legal notices that can be sometimes interpreted as entities. Therefore some manual cleanup of books is required to achieve the best results.

As mentioned in the previous section, two authors were contacted who are writing their books. They have been kind enough to give access to their unfinished books so that they could evaluate them and assess if the visualisation tells them valuable information about their work.

3 Architecture

The tool's implementation comprises the client-side, which uses *Android* Application written in *Kotlin*; the server-side, which processes and stores processed books using the *Ktor* framework; and the Continuous Integration pipeline using *GitHub* Actions and *Docker*.

3.1 Android App

The Android application is written in Kotlin. The reasons for using Kotlin for this project are as follows: Built-in Null pointer protection, Less templating work compared to Java, and Interoperability with Java Code, allowing to use of all Java libraries.

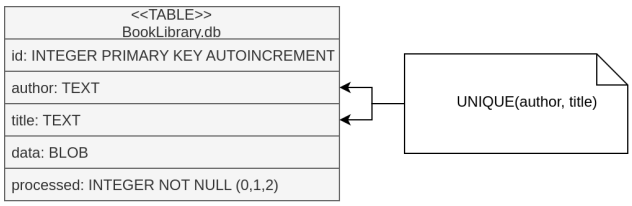


Figure 2: Simple SQLite DB schema for storing books

SQLite was used as a local database to store local versions of books. A simple Database Schema was used not to overcomplicate the development (Figure 2). The schema only uses id, author, title and processed status as fields and stores the rest in the data BLOB (Chapter text and processed information, like characters and their mentions). The use of BLOB significantly simplified the development by serialising internal class, as it allowed to evolve the data structure without requiring any schema changes. However, when writing data to a BLOB, there is a limit to how much can be saved in SQLite. Therefore, compression was used on the JSON string before storing it to save space and avoid issues with extensive data. In the future, to avoid any difficulties, it is recommended to store Book data such as chapters and text in a separate BLOB from processed data information (such as entities mentions). This helps reduce space if the BLOB limit is reached even with compression. An alternative method would be to store JSON data in a file and point the database to it.

3.1.1 App Architecture

When working on an Android app, you quickly realise that you can't just have all UI, Logic, Database and Networking code running within one Activity. It can quickly become busy, complex and make any refactoring difficult. One way to improve code testability, maintainability and extensibility is by using Model-View-ViewModel (MVVM) design pattern (Figure 3). That allows separating UI from Business and Data Logic.

It consists of the following:

- **View** - is responsible for UI, responding to user clicks. However, doesn't interact with business logic or data directly. Only calling it through ViewModel
- **ViewModel** - is located between the View and Model layers. And responsible for the business Logic. It provides communication between the two.
- **Model** - has control of the data. Can have multiple repositories responsible for interacting with Local storage, API or others.

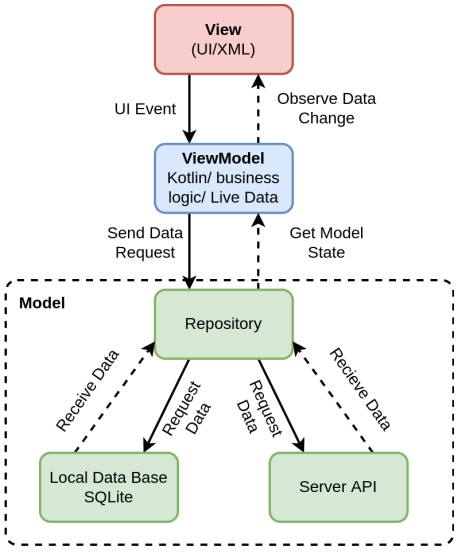


Figure 3: MVVM pattern used in Android app

Those modules don't have references to what's above them. This makes it easy to test. Additionally, they don't pass data directly between each other. Instead, they use **LiveData**. **LiveData** is an observable data holder class that allows components to observe changes in data and update UI elements.

Using this architecture initially took some time to understand and implement. However, using it made adding and extending functionality effortless and significantly sped up the development. Dependency injection was used to simplify the whole architecture and prevent repeating code, making testing much more manageable. Factory patterns were used to build dependency injection classes to simplify the process further (Figure 4).

```
fun provideMainActivityViewModelFactory(application: Application)
: MainViewModelFactory {
    return MainViewModelFactory(application, BookRepository.getInstance())
}
```

Figure 4: Factory pattern used to simplify dependency injection for MVVM

The data gets preprocessed on the phone; it reads EPUB, removes all HTML tags using the *Jsoup* library and saves it locally for further processing on the server.

3.1.2 Visualisation

Many Android visualisation libraries could have been used for visualisation. However, most of them only offered simple APIs, which weren't customisable enough, and would significantly limit project visualisation capabilities. Therefore it was chosen to go with the *D3.JS* library for visualisation as it's widely considered a mature library for visualisation.

However, the client frontend is an Android application and doesn't use JavaScript. To get *D3.JS* working within Android, WebView was used. This allowed HTML webpages to be embedded and the execution of JavaScript functions from the app. This part is considered the most fragile part of the architecture due to indirect communication between Android code and WebView. When Android communicates with a webpage, it can't call a function directly but rather executes JavaScript through text passed into the webpage (Figure 5). This part of the implementation would have to be rethought if this project was done again. Nevertheless, working with *D3.JS* was a good decision and, therefore, worth the hassle with WebView set-up, as *D3.JS* gave enough flexibility to produce the desired visualisation.

```
GraphPage.loadUrl("javascript:makeNetwork($charactersJson, $chapterNumber,
    $distancesJson)")
```

Figure 5: Android passing JavaScript function call through string

When it came to visualisation data, there was a balance between precomputing everything on the server and processing locally. Preprocessing on the server benefits from making visualisation very straightforward on a client. Furthermore, it almost entirely offloads all the computation from the phone (NFR8). However, it introduces far more issues. If everything is precomputed on the server, the size of the processed data BLOB becomes too large, to the point that even compression can't help reduce it to prevent SQLite throwing errors. In addition, it will take up space on a device (NFR8). This, in turn, will take up more network bandwidth due to higher packets size. Finally, an aspect that only matters during development. When working with visualisation, you tend to make many changes very quickly, and you require instant feedback to see how it looks. If most of the values are precomputed on the server, a simple change to visualisation data would require recompiling the entire server and processing a book every time you want to change something.

Therefore, the final design only calculates what's necessary (most computationally expensive) on the server and does some light post-processing locally. This approach allowed to keep the size of processed BLOB low, improving development workflow and still offloading most of the computation to the server.

For the visualisation breakdown, please refer to Section 6.

3.2 Server

The server is written in Kotlin and uses Ktor, an asynchronous framework for creating microservices.

3.2.1 Framework

Ktor framework was chosen for its simplicity and use of Kotlin, which was crucial. As mentioned in the previous section, there was a process of finding a balance between processing locally and server-side. Having frontend and backend written in the same language made this process simpler without needing to rewrite code in a different language.

However, due to Ktor being a new framework, the documentation is relatively vague. Because of that, hours have been spent figuring out some specific syntax required for the feature to work. An example would be when combining SSL functionality and dependency injection (For the database and connection to other services) has wasted a long time figuring out on my own, as there were no proper resources at the time of writing that explained this. Adding this functionality was crucial as it helped improve the server's testability with mocking. Moreover, there were other examples where time was spent battling with the framework rather than coding. However, overall, the use of Ktor was a good choice for its use of Kotlin and simple API.

3.2.2 Hardware

The server is being run on the Virtual Private Server (VPS), which has the most basic characteristic: 1 vCPU, 2048 MB of RAM and 50 GB of storage.

As the server will be executing some CPU and RAM intensive tasks, testing was needed to verify whether it was able to handle it. This was done by running a CoreNLP library with test book text. Due to "Out of memory error" the process would always shut down for large texts. This made sense as when reading CoreNLP documentation, it mentioned that it loads its models into RAM. As server had limited resource of only 2 GB RAM, it can easily take up all the resources. An easy fix would be to upgrade to a higher RAM capacity server. However, this is so much more expensive and unnecessary.

Therefore to prevent this from happening, SWAP space was added (10 GB was used, which was more than enough for the application), which instantly solved the issue. Now, when loading a large model into memory, some of it could be stored in the SWAP.

State	RAM	SWAP
IDLE	169 MB	0 MB
IDLE + Running Server	400 MB	0 MB
Processing Book	1790 MB	1310 MB
IDLE + Running Server After Processing	820 MB	2100 MB

Table 2: SWAP used to reduce the RAM usage of the server

Initially, the whole server set up only takes up about 400 MB, but when using CoreNLP the usage of RAM skyrockets (Table 2). The size of books has an additional effect in increasing this usage. However, SWAP space helps to prevent server crashes. Additionally, there is no dramatic decrease in performance due to the server using SSD.

3.2.3 Dockerisaiton

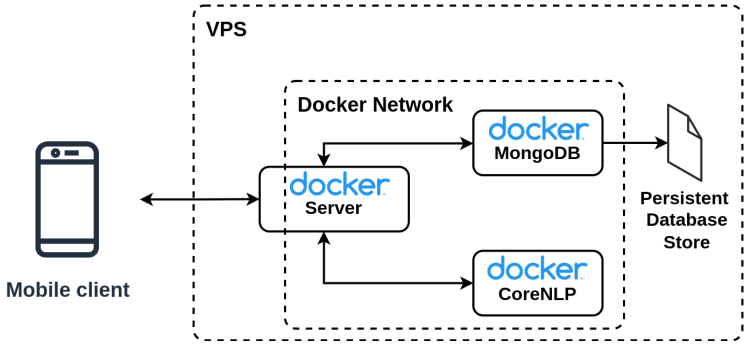


Figure 6: Server Docker set-up

To improve modularity, security and ease of server setup, the whole server is running within Docker Containers, which means that the Ktor server, MongoDB and CoreNLP are all running in independent containers with a Virtual Network between containers which improves the server's security (Figure 6). The dockerised server helped during development as it could quickly be run under any OS. This setup protected the server during a security incident. Please read more about this in Section 4.

3.2.4 Storage

To reduce the computational overhead of processing the same book multiple times, the *MongoDB* database was implemented to store processed books and keep track of books that have failed to process. So, once the book has been processed, it can simply be retrieved from the database. Similarly, if the book fails to process, the server won't try processing it until the server admin removes it from this list.

NoSQL database was chosen because it allows rapid change of the underlying data structure without constantly redefining its schema. Because book data was going to be stored as JSON

regardless, this allowed to quickly store and retrieve data without needing to assemble it before sending it off to the client. The book itself isn't being stored, only storing its processed data to avoid violating terms and conditions (NFR3). This also came in handy when the server was compromised, as no valuable data could be stolen.

3.2.5 Processing

When using CoreNLP, there are many annotators available, which is one of the reasons this library was chosen. Turning all annotators to get the "best" and the complete picture might be tempting. However, the performance can suffer due to the text sizes being processed. Therefore, some tests have been performed to decide what will be used for the final system.

Two particular processing pipelines have been tested, the first just uses NER Tagger, and the second uses NER Tagger + Coreference Resolution. It was also interesting to find out if processing books by chapter would improve performance compared to processing the whole book at once.

Coreference resolution allows getting extra information such as pronouns and which entity they belong to. This way, if the character is only mentioned by name once in the text but after is referred to by pronouns, the precision of the mention count can be increased, giving further help in determining the interaction value between characters.

Books of various sizes were processed in 4 different ways:

- By Chapter NER Tagger
- By Chapter NER Tagger + Coreference Resolution
- Whole Book NER Tagger
- Whole Book NER Tagger + Coreference Resolution

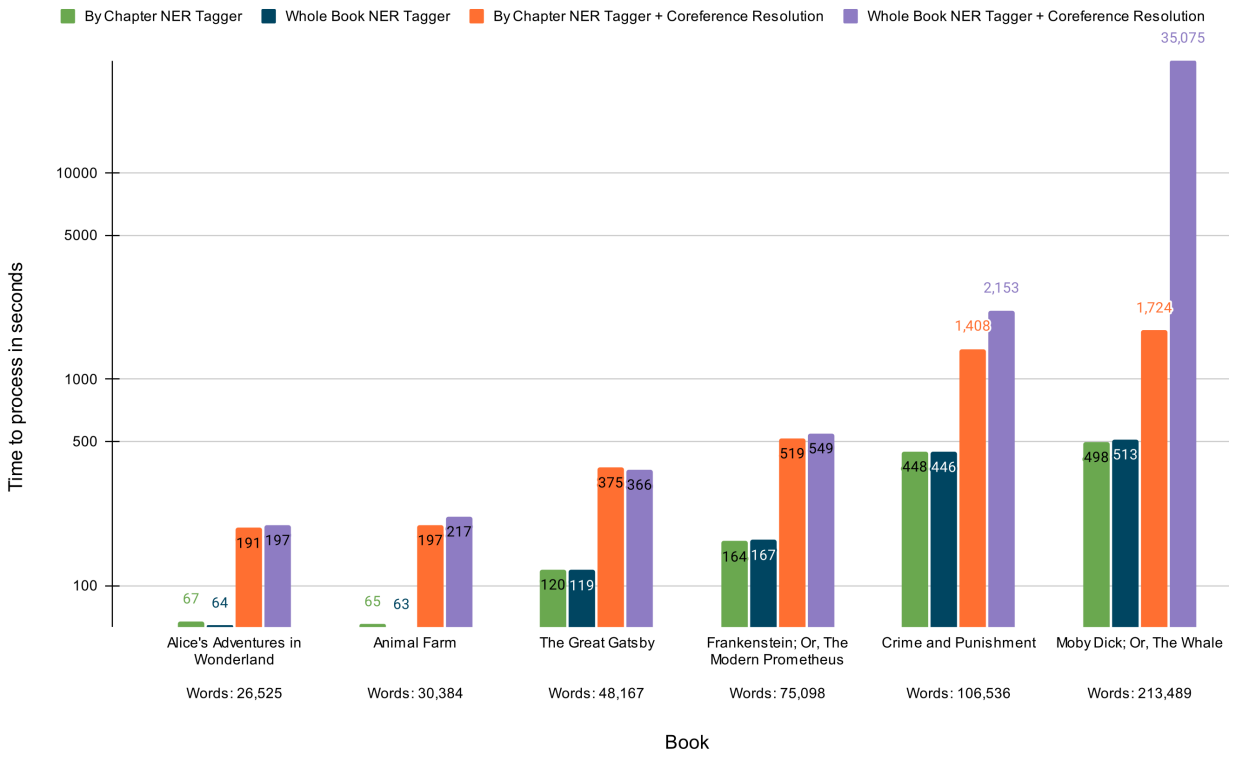


Figure 7: CoreNLP performance testing for NER Tagger and NER Tagger + Coreference Resolution. The test was run on a machine with i7 and 24 GB of RAM

If we first look at by chapter and whole book processing for the NER Tagger in Figure 7, we will see hardly any differences that this produces. There are almost no improvements in performance when it comes to processing relatively small books; as the books get longer, the gap between both performances doesn't increase dramatically. Therefore, there is no point splitting up books into chapters when processing just using NER Tagger, as this doesn't improve performance for even a relatively large book.

When processing with NER Tagger + Coreference Resolution, we can see that it increases processing time by more than 3 times compared to NER Tagger processing when using By Chapter processing. Up until Frankenstein, both by chapter and whole book NER Tagger + Coreference Resolution processing stays relatively similar. When we get to Crime and Punishment, we see chapter processing benefits. For a bigger book such as Moby Dick by chapter processing starts outperforming whole book processing by literal hours. According to CoreNLP (*Understanding memory and time usage*, n.d.), this is due to Coreference Resolution being very sensitive to total document length since they are quadratic or cubic. Therefore, because Coreference Resolution is highly sensitive to large texts, it's recommended to split the book into chapters to speed up the processing and not consume as much RAM. However, If the book is split into multiple chapters while improving processing time, character coreferences will have to be joined somehow together across different chapters. Additionally, Coreference Resolution also requires a lot more RAM - about 8 GB compared to around 2.5 GB when using NER Tagger.

From those tests, we can conclude that Coreference Resolution is far more computationally demanding and greatly depends on the size of the document. However, we can still get excellent performance from Coreference by splitting books into chapters. Nevertheless, Coreference Resolution failed to produce satisfactory results, as when examining returned data, it would often tag wrong parts of the sentence as the reference to characters. Due to literature works usually being too large, it degrades the quality of the results. In addition to it, this also added the need to do some further post-processing in joining character and location entities into one entity between chapters. And finally, due to it not producing great results and still taking 3 times longer than NER tagger, it wasn't a great choice to use it. Therefore, only required annotators were used, **"tokenize,ssplit,ner"** for NER Tagger.

3.3 Networking

After choosing the CoreNLP pipeline, to verify the performance the system will expect for networking in terms of the timeout; further testing was performed with CoreNLP + extra Server Processing (chapter, distance) to determine the wait time (Appendix C).

Initial network architecture relied on a timeout. Thereby sending a GET request with a book in the body and waiting for the response with processed Book JSON from the server. However, testing demonstrated that it would take up to 7 minutes to process a book at times and take even longer if multiple books are being processed simultaneously. It wasn't a great idea to make a user wait for this long with their phone open and not switching their app (NFR1).

Therefore the process was redesigned (Figure 8). It performs POST requests by sending the book for processing and getting a confirmation. And then, the app periodically does GET requests to check if the book has been processed.

This way, the user can close their app (or even delete and reinstall the app on a completely different device), and once the book has been processed, they can simply just retrieve it from the database. This design additionally allows other users to benefit from processed books.

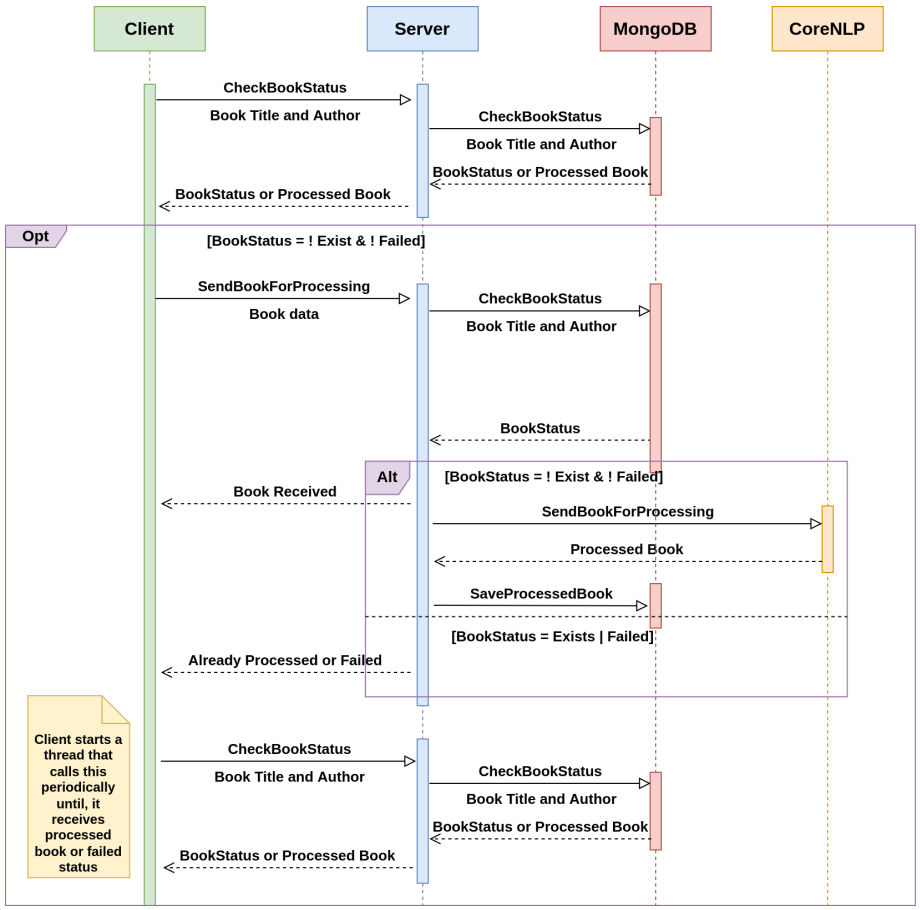


Figure 8: Network API for book processing

One current problem evident from this graph is that no processing queue is implemented here. While the book is being processed, the server has no record of it being in the pipeline until the processing is done or failed. This wasn't considered a big problem, given that this architecture was more a proof of concept. However, this should be implemented in the future if this system is going to be used in a production environment.

3.4 CI Pipeline and Testing

For both the Server and Android, unit testing was used extensively (Table 3). Using a lot of dependency injection to simplify testing and enable easy mocking of functionality.

	Class, %	Method, %	Line, %
App	81.6% (31/38)	69.8% (60/86)	60.6% (189/312)
Server	39.4% (13/33)	52.5% (31/59)	67% (252/376)

Table 3: Unit testing of App and Server modules

Some modules couldn't be tested, like database controller or network calls, as testing them would be impractical or would just be testing a library (Hence the low values in server test-

ing). Nonetheless, the main functionality has been tested in both Server and Android.

As the Android application has UI, it makes sense to perform end-to-end testing. However, recent changes to the testing API and Documentation not fully reflecting all the changes caused a lot of ambiguity and problems getting UI tests working. For example, ActivityTestRule (*ActivityTestRule*, n.d.) is now deprecated and superseded by ActivityScenario (*ActivityScenario*, n.d.). However, at the time of writing, there was no proper documentation on how this can be used for intent mocking, which is a fundamental part of end-to-end testing on Android. Due to these issues, it was decided to abandon this and continue with other features, relying on the unit tests. Unit tests can still ensure a stable user experience, as long as some manual UI testing is performed. This wasn't a problem as the application UI was fairly simple. Although it should be mentioned that when documentation gets fixed, end-to-end testing should be implemented, as it offers a quick way to test application performs as expected programmatically.

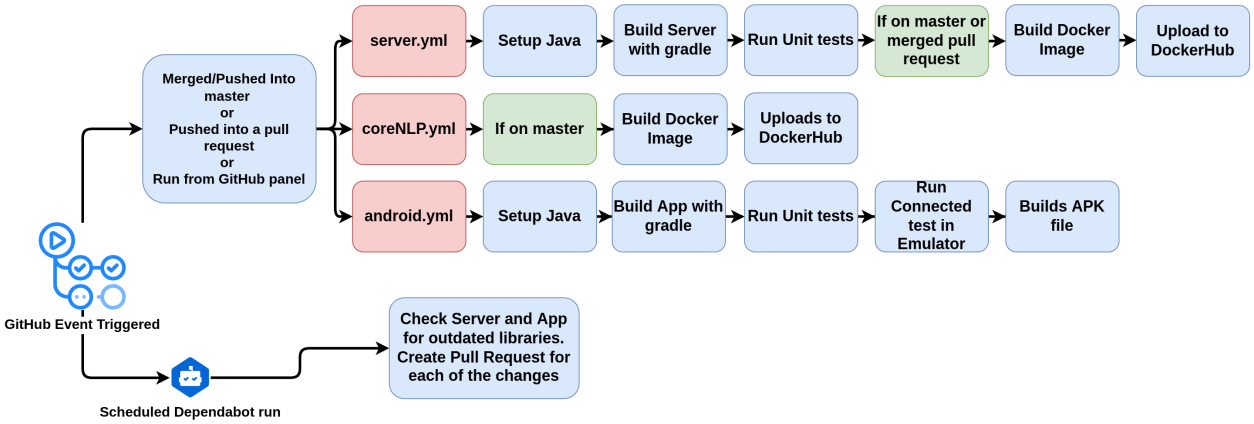


Figure 9: Continuous Integration Pipeline

To maintain a high standard and simplify development, a Continuous Integration pipeline was built with *GitHub* Actions (Figure 9). When committing or pushing changes, it would automatically run tests. When committing to master, in addition to tests, it would also build docker images for CoreNLP and Server and push them to the image repository, where they could easily be downloaded with a simple command line on the server.

It can be hard to maintain all the application's libraries. Dependabot was used in the project to simplify this process. It periodically scans the repository to find upgraded libraries in specified projects. Then it automatically creates pull requests with required changes. This way, the focus could be kept on adding functionality, but still, keeping the libraries up to date. A testing pipeline allowed the tests to run automatically whenever Dependabot created a pull request, telling if there were any breaking changes when upgrading this library.

4 Security Incident

In this section, a security breach incident will be reviewed, explaining a possible way the attacker exploited the technology stack and describing how those vulnerabilities have been patched.

4.1 Vulnerability

Although security was never the project's focus; it is still worthwhile to touch upon this. There was a security incident where the server was hacked, possibly with a NoSQL injection exploit. All Database tables were deleted and held for ransom (Please find the attacker's message left in the database in the Appendix D). Only the processed book information (entity names and appearance) was stored, not the books themselves. Attackers couldn't have gained anything from this exploit. This data can easily be generated again.

Before the incident, it was known of the server's lack of security, especially user input sanitisation, as it was only planned to do the bare minimum to protect it from simple attacks. When reading through MongoDB documentation, it give a false sense of security when using BSON objects for MongoDB queries (appendix E). Even though it states it's only protecting from traditional SQL attacks, it was still assumed that the server should not be vulnerable to injection attacks.

According to (Egor Homakov, 2015) in the Ruby Implementation of MongoDB, a BSON exploit allowed for injection, allowing arbitrary execution of code. Although it might have now been patched. The project uses a less popular combination of MongoDB and Kotlin backend with KMongo client API. Therefore there might be some other vulnerabilities that haven't been able to find to reproduce, to test the server.

An additional issue of MongoDB is that by default, it's allowing JavaScript execution directly on the server within the query, which might have been used with BSON Injection to gain total control of the whole server. Fortunately, my server was dockerised, with a separate virtual network for communication between containers. Due to this level of security, only the Database was affected and nothing else, as far as the examination showed.

Since there was no other more obvious way for the attacker to gain access to the database, as there are only two ports open, 8443 for communication with the Ktor server and port 22 for SSH connection. NoSQL injection seems like the most likely attack vector.

4.1.1 Patch

Now, having a rough idea of how the server was compromised, the patching could be applied. The first thing was to implement user input sanitisation. However, there is no native Java MongoDB Sanitisation that could be found. The only viable solution was *mongo-sanitize* which was incompatible with the server framework. Therefore a custom input sanitisation

was implemented. Which only allowed the following symbols in the user input **[a-zA-Z ;':,.0-9]**. This can drastically reduce an attack vector. Additionally, the user input was limited to just 40 characters, making it significantly harder to exploit vulnerabilities due to having less space to work with.

JavaScript execution was also disabled within the docker file. Further, a security token now was required to communicate with the server for all requests. Because the application was still in development, the server only allowed connections to the server from the University Campus IP range.

From this incident, a couple of lessons have been learned. New technologies might have issues with vulnerabilities from untested APIs, unreasonable default settings and unclear documentation, which gives a false sense of security.

5 App UI

The application UI is relatively simple (Appendix G). It consists of the main page where the user can upload new books and review books already uploaded before. As well as, seeing the processing status of all of the books and whether there is a connection to the server. The book can also be deleted from this screen.

Selecting one of the books will open a reading activity page, where the user will be able to scroll through the book and switch between the chapters. There can also be seen buttons for Characters and Location, which will show an alphabetically sorted list of entities that can also be clicked to get to the entity profile.

From the Reading activity page on the top right corner, there is a three dots menu from which the type of visualisation can be selected. The visualisation page has zoom functionality, which allows the user to examine the visualisation closer.

5.1 Highlighting Entities in Text

As a different form of visualisation, text highlighting of entities was implemented. This allows users to see all the characters and locations within the text while reading. Characters and Locations are shown in different colours (Figure 21). Each entity is clickable, showing the character profile of the entity, with how many times they appeared in each chapter (Figure 24).

When shown to the authors, they found a character they forgot to rename or forgot existed in this chapter during the testing of their books.

6 Literature Visualisation

In this section, we will look at visualisations and discuss their effectiveness in being alternative summaries. This section will also discuss possible future improvements and lessons learned.

6.1 Location Visualisation

Location wasn't used for visualisation, as during interviews, Authors and Readers showed more interest in the characters than locations. They were interested in location only if it was used together with characters. In addition, the data coming from the CoreNLP wasn't as robust for the location as it was for the characters. This check was performed visually by looking through the list of tagged locations, which contained many imperfections.

6.2 Representing Character Appearance

6.2.1 Pie Chart

First visualisation attempts to represent the proportion of each character mentioned within the book or chapter. In this seemingly simple visualisation, the goal is to represent character appearances in the text and allow the user to compare it with other characters to get a feel for the proportion of the story taken up by a particular character. This visualisation should enable the user to interpret who the main characters are, who the supporting characters are, and who are just background characters.

To represent the appearances of a character in the book can be simply done by counting the number of times the character appears within the text. Although Pie Chart seemed like an excellent way to represent character proportions, in practice, it gets very messy with real data and is hard to read.

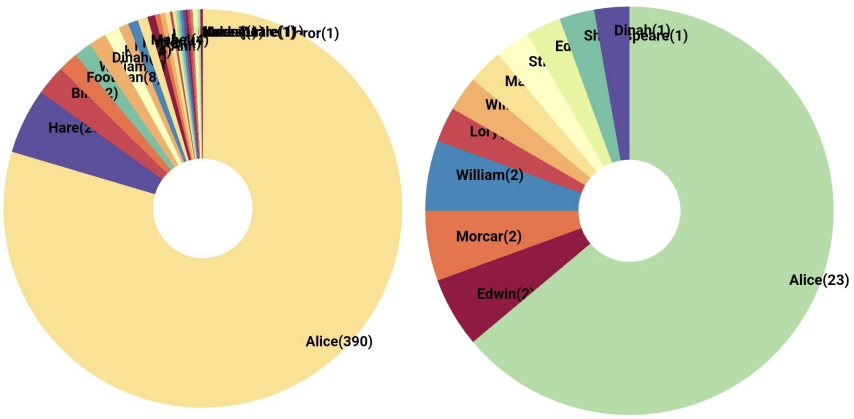


Figure 10: Left: Whole book, Right: Chapter 4 for Alice in Wonderland.

From the book representation in Figure 10, we can interpret that Alice is the main character as she appears more often than others. We can also see that Hare often appears in the whole

book, which might tell us that they are a supporting character. Those who appear only a few times can be classed as background characters.

However, this visualisation has the following issues: the main character takes up too much space without adding more valuable information. It leaves very little room for the rest of the characters. If the character appears only a few times, it is hard to see that character. Only displaying the top 3 characters and adding the rest into the "other" category could solve this, but this obstructs information from users. Adding leader lines or legend would be the best way to make this visualisation more viewable, but this can still be hard to see when you have many characters (Ex, The Great Gatsby).

An interesting point was made during the interviews. Readers were most interested in the main characters (Most appearances), and Authors said they tend to look at minor characters (Appendix F). Therefore, the visualisation should be able to represent minor and major characters simultaneously.

According to (*Pie Charts in Data Visualization- Good, Bad or Ugly?*, 2021), a pie chart is not the best way to represent this type of information. Pie charts only really work for data that consists of 2 or 3 options. They also make it very hard to perceive information as it relies on our eye measurements. A good alternative is a bar graph or a lollipop graph.

6.2.2 Lollipop Graph

A Lollipop graph was implemented as an alternative for the Pie chart (Figure 11).

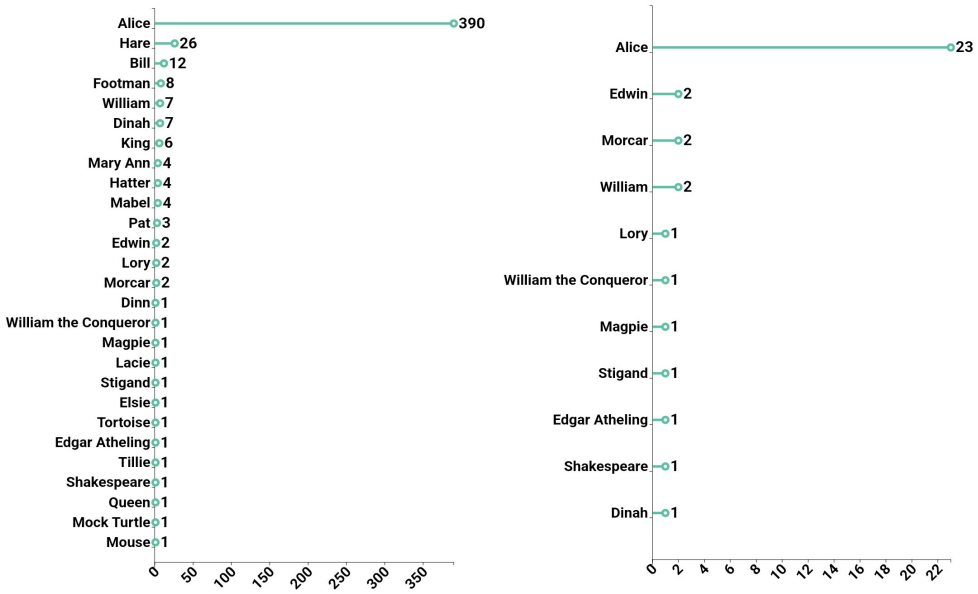


Figure 11: Left: Whole book, Right: Chapter 4 of Alice in Wonderland.

With Lollipop, we can instantly see the major and the minor recurring characters as well as allowing us to represent different numbers of characters consistently. It retains Pie chart

properties by visually representing proportions with a horizontal bar. This allows comparing other characters by the length of their lollipop and the numerical scale.

6.3 Character Network Graph

The second visualisation is Character Network. This visualisation shows the interaction between characters. Users should be able to interpret the importance of the character and get insight into with whom the character was interacting.

6.3.1 Visualisation Implementation

First, a visualisation was designed to represent the interaction between characters (Figure 12). When thinking of what graph to use to represent character interaction, rather than the interaction between character A and character B, this interaction needed to be represented for all characters simultaneously to get a bigger picture. Therefore Force-Directed Graph was used as a base for this visualisation. It allows nodes to represent characters and links to represent the interaction between those characters.

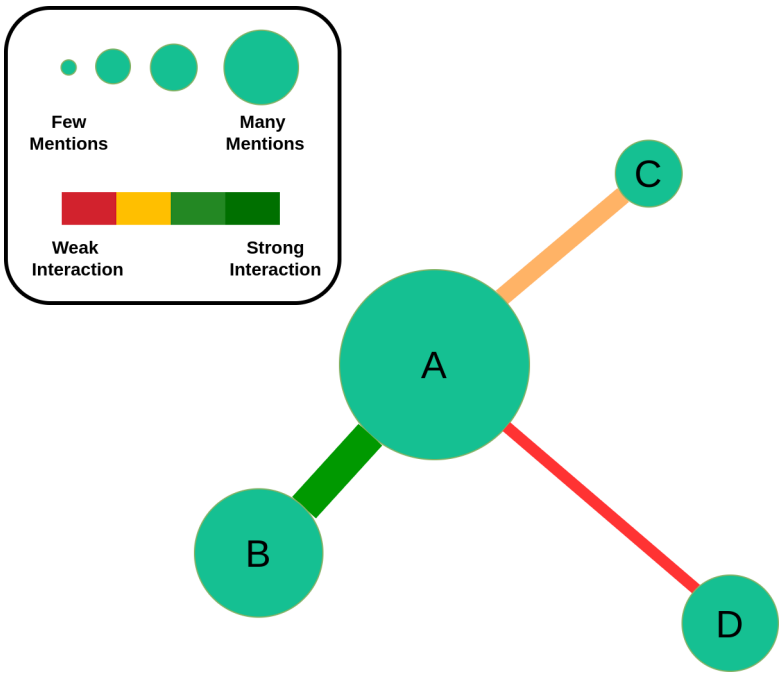


Figure 12: Character Network diagram for representing "interaction" between characters

There is also a negative charge between nodes to spread them apart. The physical pull is also present, but it's not as strong in visualising any large differences. Raw distance could not be used to define colour and width, as they drastically differ between chapters. Therefore they had to be mapped to a domain and range to allow it to be used to define colour and width. With this simple visualisation from the Figure 12, we can interpret that character A appears the most and character C the least, from the size of their nodes. From the properties of the links, we can see that A and B have the most "interactions" among those characters.

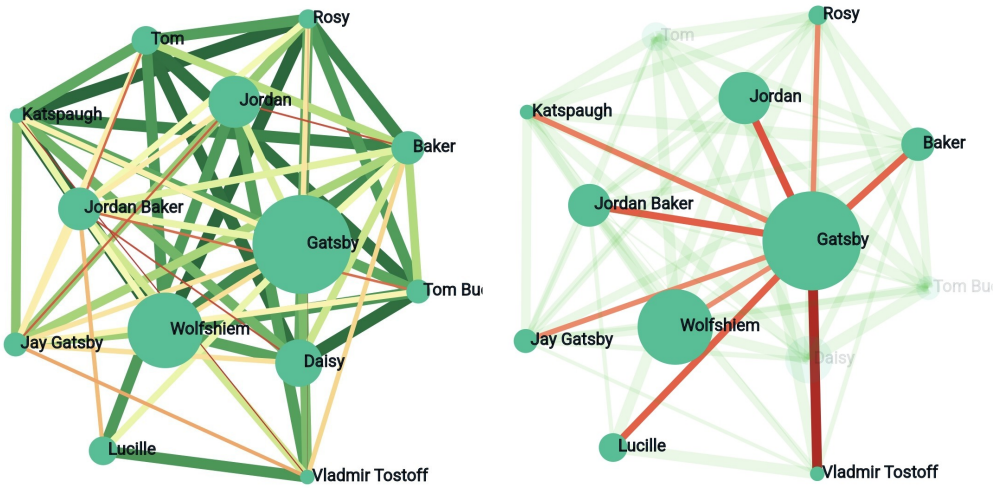


Figure 13: Focusing on a character by highlighting character links. The Great Gatsby chapter 2 at 74% top Links and 15% top Characters

However, with more complex visualisations, it can become hard to see each connection clearly. To improve visibility when clicking a character, it will only highlight links and characters related to the clicked character (Figure 13). Opacity is not reduced to zero to help keep everything in context; therefore, focused links change their colour scaling from (Green to red) to (Strong red to faint red) to stand out. This way, the user will be able to focus visualisation only to highlight what they are interested in.

There is a negative charge between nodes to spread them and pull on the links to bring nodes closer together. However, optimal values couldn't be found for pull force during testing to make it work with all books. Therefore the pull force was reduced, and a minimum link length was set to prevent nodes from clustering together into a single point.

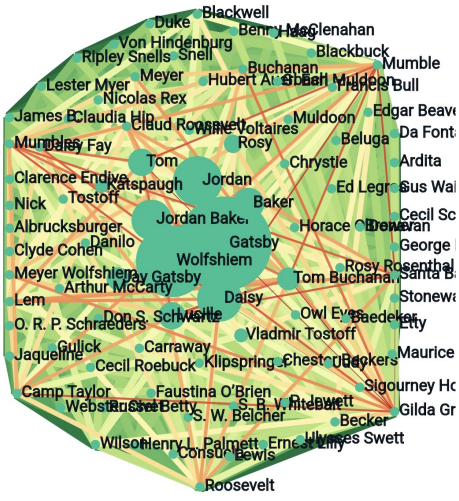


Figure 14: The Great Gatsby chapter 2 at 100% top Links and 100% top characters

Different books had different numbers of characters. Some have relatively few characters, and The Great Gatsby had 89 characters in just chapter 2 (Figure 14). It gets too crowded to make out individual connections. Even using the focus feature doesn't help in this. To solve this, the number of characters and links needed to be reduced. Manually setting it would only work for a specific book and would have to be set in code in advance. Therefore two user controls were given (Figure 15): The first control allows to display top largest N% of characters by mention, Second displays the top shortest N% of links. This allows the user to choose the amount of information available rather than programmatically obstructing it.

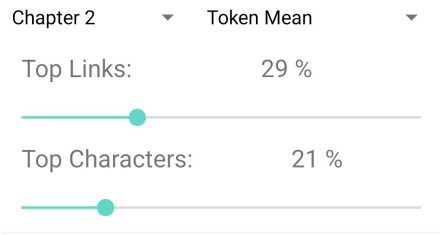


Figure 15: User controls for customising visualisation

6.3.2 "Interaction" weighting

Now, let's define the "interaction" between characters and how it's calculated. Interaction weighting should define how much characters interact with each other. Several methods were implemented, and more advanced methods were identified for finding "interaction".

Token Average

The most simple heuristic involves counting the number of Tokens between entities. This does not directly tell us of interaction between characters, but it can give us an idea. Because logically, characters that interact with one another should appear close within the text.

We get this distance by counting the number of tokens between mentions of each character and averaging them.

$$TokenDistanceBetween(A, B) = \frac{\sum_{i=0}^{A.size} \sum_{j=0}^{B.size} |A[i] - B[j]|}{A.size * B.size}$$

Where:

- **A** and **B** are arrays of token locations of characters

Punctuation Average

Token Average can give us an idea of how close characters are, but it's not perfect. Some sentences can be long, and others can be concise. The length of the sentence doesn't always translate to a longer logical distance. To better tell the distance between two entities, punctuation can be used. Punctuation is a defined way of telling when an idea ends and when a

new one begins. This way, an overly descriptive writing style doesn't affect the interaction weight much.

End Of Sentence	**. ? !**
Parts Of Sentence	**, : ;**

Table 4: Two punctuation groups used for calculation

There are two separate groups of punctuations that are used for calculations: End of Sentence and Parts of Sentence (Table 4). Other punctuations aren't used as often within literature or don't add more information (Figure 16). Therefore those are not used to reduce storage requirements (NFR8). Despite the high frequency, quotations are not used in calculations, as removing quotes didn't make much difference in my observation. However, if not faced with storage limitations, it would be worth using them. This was the sacrifice that had to be make due to storage limitations.

Punctuation	Frequency
,	2412
" and "	2230
.	969
!	451
:	230
?	202
;	192
-	138
*	60

Figure 16: Top 9 punctuations from Alice in the wonderland frequency table.

Each of them has a different weighting assigned to them. End of Sentence has a higher weighting than Part of Sentence. Those weightings are applied individually on client device just before it gets visualised.

$$PunctuationDistanceBetween(A,B) = \frac{\sum_{i=0}^{A.size}\sum_{j=0}^{B.size} CalculatePunctuationsBetween(A[i],B[j]))}{A.size * B.size}$$

Where:

- **CalculatePunctuationsBetween** counts punctuations from Table 4 in the text between two character mentions

There was a problem distinguishing between links represented by colour and width in visualisation for both Token and Punctuation average. The values were tough to distinguish because it wasn't using raw value but rather scaling it down to fit into the small screen. An exponential of two was used for a non-linear response, making it easier to distinguish larger and smaller values.

6.3.3 Character Network Evaluation

When visually inspecting visualisation with Token and Punctuation average, there wasn't much difference, only subtle width and colour change. The overall structure wouldn't change much, mostly due to physics being too weak to show small changes in interaction values.

Token Average				Punctuation Average			
Raw: Min:42.25 Max: 3048516		Scaled: Min 0.5 Max 25		Raw: Min 25 Max 1159929		Scaled: Min 0.5 Max 25	
Characters	**Rank**	**Raw Distance**	**Scaled Distance**	**Characters**	**Rank**	**Raw Distance**	**Scaled Distance**
Alice,Shakespeare	25	339686.27	3.22	Alice,Shakespeare	28 -3	120409.00	3.04
Alice,Dinah	30	505953.83	4.57	Alice,Dinah	30	212721.49	4.99
Magpie,Alice	31	630712.16	5.57	Alice,William	31 +1	217520.85	5.09
Alice,William	32	685692.05	6.01	Edgar Atheling,Alice	32 +1	220981.76	5.17
Edgar Atheling,Alice	33	693671.71	6.07	Stigand,Alice	33 +1	264643.14	6.09
Stigand,Alice	34	835713.89	7.21	Magpie,Alice	34 -3	272665.60	6.26
Lory,Alice	36	886463.16	7.62	Lory,Alice	35 +1	291130.62	6.65
Morcar,Alice	37	907463.34	7.79	Morcar,Alice	36 +1	300780.71	6.85
Edwin,Alice	38	910281.95	7.82	Edwin,Alice	37 +1	300780.71	6.85
William the Conqueror,Alice	39	1014661.99	8.65	William the Conqueror,Alice	38 +1	348767.27	7.87

Table 5: The table shows the distances between Alice and other characters within chapter 4 of Alice in the Wonderland. The RAW distance is the actual distance between characters after being raised to exponent 2. Scaled distance is done to get a uniform graph across different distance methods. Alice was chosen as the focus because she is the main character.

When comparing those two methods in Table 5, we see very little difference in their order, only changing slightly. Therefore those methods can get us some rough estimates of the character interactions. There is not much difference between both methods as they give similar descriptions, just with different numbers. This might be because authors are consistent in their style, which might mean that token and punctuation distance will be proportional within the same work.

An issue with those methods is that it's averaging values. This can be evident when looking at the main character (Figure 17). As the main characters tend to be present throughout the chapter, other characters sometimes only appear within specific portions of the chapter. By averaging, the value won't reflect that both characters were close at the beginning of the chapter. This could be solved by only getting the distance between entities within one paragraph, which is not possible in this case due to the dataset not having consistent paragraph break methods. A more straightforward method would be to take mentions within a certain number of tokens, reducing the effect of extremes. However, it might be hard to choose a distance limit programmatically. The value could be selected manually on the client, but this moves more computation to the client device (NFR8).

6.3.4 Improvements to Current Approaches

There are several improvements and recommendations that can be made for the current approaches for the character network.

The biggest issue with the current approaches is averaging distances, which distorts the "interaction" value. Clustering character appearances could improve finding distance between characters that is not affected by extremes. As interaction value could be calculated between clusters rather than raw mentions. Because there are generally fewer clusters than the original number of mentions, this can allow performing computations on the device. The user can control how far each cluster has to be from the other to be taken into the calculation. Token or Punctuation Distance can be used as a distance metric between clusters, based on the user selected parameters. It would also be interesting to see how cluster overlap could be used for interaction value calculations.

This was tested with Mean and Median Token to get a single location for each character and calculate their distance (Figure 17). As it only produced a single point, it wasn't perfect. However, you could see improvements to the main character distances, as it now shows connections to less frequent characters.

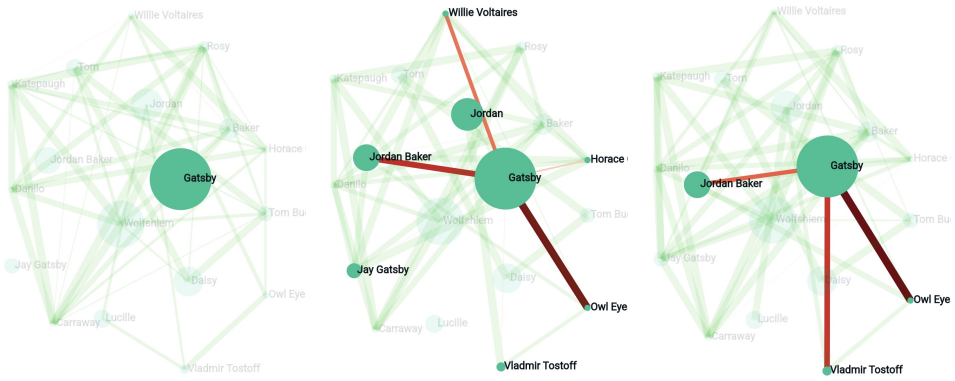


Figure 17: Left to Right: Token Average, Mean, Median. The Great Gatsby Chapter 2 top links 29% top characters 21%

For entity clustering a method such as K-means can be used. However, a number of clusters need to be known beforehand. The Elbow Method could be used to find this. Jenks natural breaks optimisation could also be used, as we are working with 1D here. A different good approach is using Kernel Density Estimation (KDE) as it doesn't need cluster number as an input (Matioli et al., 2018). By taking token locations of all character mentions, we can make a 1D array that will be clustered by KDE, with maxima and minima (Figure 18).

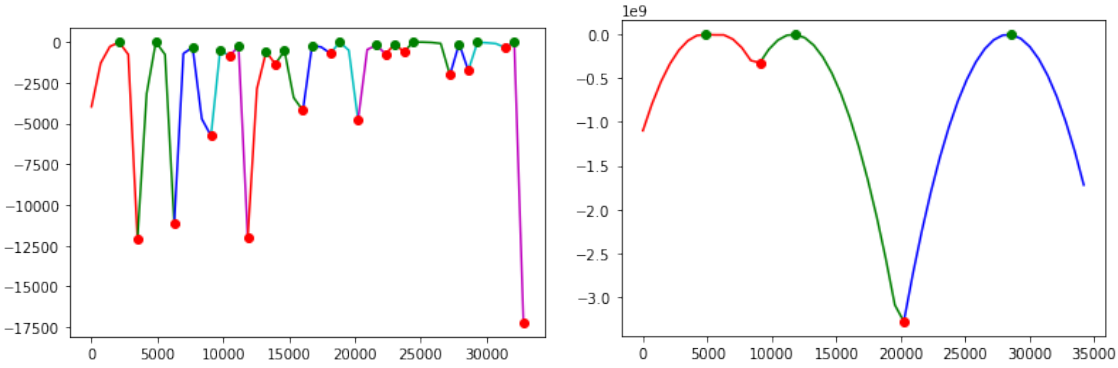


Figure 18: From Alice in the Wonderland. Appearances of Alice (Left) and William (Right) in the book. William appears in chapters 3, 4, 6, and 11 and Alice appears in all chapters. The x-axis represents the token location within the book. Possible clusters are shown with colour.

An additional heuristic for determining "interaction" could be quotes. If a character mentions the other character's name, the distance could be reduced by a percentage to reflect a stronger link. Coreference resolution can be enabled within CoreNLP to find speakers. However, it is very computationally expensive. Therefore to find the quote owner the closest named character outside the quote can be assigned.

Overall, all those heuristics should be used together to calculate the "interaction" value. Using Clustering to find distinct regions of character appearance, using quotes to reduce the current distance to reflect on a stronger interaction.

6.4 Visualisation Conclusion

The hardest part wasn't the visualisations themselves, but the aspect of making it easily viewable on the small screen.

By working with the users together, the Pie Chart visualisation was fixed by finding an appropriate alternative that satisfied both readers and authors. When it came to major or minor characters, they wanted the exact opposite information to be displayed. By talking to all groups, a Lollipop graph was produced that satisfied everyone and made visualisation easier to view.

In terms of Character Network, the users enjoyed the design of the visualisation and controls (especially the highlighting feature). As it made visualisation more dynamic and customisable. However, the testing found that the methods used to find interaction values were too simplistic and didn't estimate character interactions well. Therefore, more advanced methods were proposed to give users even more control and improve interaction value estimates.

During the final project inspection, it was noted that the text was too small to read. The font size was increased for all visualisations and made bold to improve readability. Those

visualisations were designed to give a big picture. Therefore it makes sense why it can be hard to read some of the text. However, because those visualisations have zoom functionality, none of the authors or readers ever told about their discomfort with the font size.

Location entities weren't used for visualisation due to their quality and users' disinterest; therefore, no development time was spent on this. However, it can still be very beneficial to visualise characters and locations together to maximise information retrieval for users. Proposed visualisation can be found in Section 8.

Overall, the visualisations presented here gave users insight into books. They allowed them to tell characters' involvement within a chapter and their interactions with each other.

7 Evaluation

During the literature review (Section 1.1), it was identified that readers struggle to remember complex character names or when there are many characters. A parallel was drawn from conversation with two authors and a reader, that both groups experience similar difficulties. From this, it was concluded that visualisation might be able to offer an alternative form of book summary that could help solve those issues for both groups. A further review of current literature processing solutions identified not many easy-to-use applications that allow users without much technical knowledge and powerful machinery to use NLP techniques to get insight into their literary works, especially visualisation. The evaluation of the visualisation was partially performed throughout the report; please refer to those sections (6.3.3 and 6.4) for a more detailed evaluation. To evaluate the project, this section will see if requirements were satisfied from both end-users and a technical point of view.

7.1 End-User

Several interviews were conducted with the authors and readers during the early stage of the tool development and at the end of development. To verify that the tool is effective in the final interview, the authors were asked to evaluate their books in the app. This way, they could assess the tool's effectiveness relating to their works. Readers were given several books that they were familiar with.

Although there was a plan, the process was more unstructured to allow users to explore the tool independently. The demonstration consisted of:

- Load the book into the app.
- Look at entity highlighting in text, entity list, and entity profiles.
- Look at visualisations.

The authors spotted character mistakes in their books (Such as misspelt or old character names that weren't changed everywhere). Additionally, they were surprised to see that one of their characters was mentioned within a chapter they didn't expect to be mentioned. Throughout the demo, authors could be seen reminding themselves of the storylines and characters with the help of the tool. Altogether, the tool gave an interesting perspective to authors on their books. There was clear evidence of the benefit that the tool brought to the table, even during the short demo session. However, the authors didn't find the tool as useful as it could be. Their main gripes with the tool are that they feel it would be way more helpful if it gave an emotional breakdown of each character or a chronological timeline of actions a character performed. Which were noted in the author's functional requirements (AFR2, AFR3) and were marked as **won't** due to them being out of scope and the time limitations.

Readers were pleased with the application, saying that it exactly delivers on the premise of helping them remember character names in literature. They have particularly enjoyed Character Network controls and highlighting.

Overall, users were happy with the delivered tool and found useful information during the demo. Any issues they found during the demo were minor or expected (AFR2, AFR3). They were all noted down so that they could easily be implemented if further development was to happen with the app. Please see Appendix F with their feedback.

7.2 Meeting Requirements

This section will discuss whether the system has achieved its functional and non-functional requirements (1.4.1 and 1.4.2).

7.2.1 Functional

By focusing on the **must** and **should-haves**, using the MoSCoW system, a minimal viable product was delivered with all the features that were determined as necessary. Only leaving features that enhance the tool but otherwise do not offer key new functionality (RFR8, RFR9 and AFR1).

Overall, the main functional requirements have been satisfied. Knowing that there would be a lot of back and forth with the users to get the visualisation that satisfied both groups. The functional requirements weren't overly extensive given the limited time available. The remaining requirements should not be hard to add, given the extensible app architecture, processing and development pipeline. The Android application uses an MVVM pattern that simplifies extensibility and maintainability. The server is written with a simple Framework making it easy to add any additional routes. Both databases on the local device and the server use JSON to store data, significantly simplifying any further modifications to the processing pipeline. The developed CI pipeline allows to keep the tool libraries up to date and automatically performs tests when committing new changes to the system.

7.2.2 Non-Functional

The tool achieved its non-functional requirements. They played a more important role in shaping the tool than the functional requirements. Rather than setting out goals, they are setting limitations.

NFR1 Performance was satisfied by extensively testing the processing pipeline to find optimal parameters with CoreNLP for processing large literary books. Taking at most 6m:49s for the longest book in the dataset used. The network was also designed to take long processing time into account to minimise user frustration by allowing users to close their apps while the book is being processed.

NFR2 Reliability was satisfied by setting a timeout on the processing and saving a book as a failed processing attempt that would notify the users. Prevented resources from being wasted if the book was taking too long.

NFR3 Security was satisfied by only storing processed books, using SSL and Docker for the server. That ended up being what has protected the server during the security breach (Section 4). An appropriate security patch was implemented to remove the vulnerability.

NFR4 Localization was a limitation that helped me focus only on one language. As a bilingual, I was tempted to implement processing and UI in my language. However, that would be hard to assess my work for the project supervisor. As well as adding additional non-key functionality. Therefore this was satisfied by only keeping localisation to one language.

NFR5 Learnability and **NFR6 Efficiency** were satisfied by making a straightforward UI that allowed users to get to the visualisation they wanted with just three clicks. The users also found the application interface intuitive and didn't ask for much help while navigating the application. The only point of improvement would be to have better descriptions for the visualisation control panel, as some of those required prior explanation to the users.

NFR7 Technical knowledge The tool can just be used as an app that doesn't need any set-up to start using it. Therefore, this requirement was satisfied as users don't need any command line or coding skills to process their book. They can simply install the app and upload any standard EPUB file.

NFR8 Hardware requirements With API and Processing testing performed during development to balance processing time and complexity, applications could perform complex and long computations. Using a server for those computations allowed users to have very simple phones and still upload any book they wanted. During development, an emulator was used that runs through the emulation layer, which can significantly affect emulator performance, despite this application run smoothly without major problems.

Overall, non-functional requirements introduced a unique challenge when developing the application as they had to be taken into account when adding new functionality.

7.3 Technical evaluation

This section will evaluate some of technological stack and architecture choices made during the project.

The developed CI pipeline has significantly simplified application development by automating more mundane tasks such as testing and Docker image building. Allowing to concentrate on the feature work of the application.

MVVM architecture in Android applications has made it easy to add new functionality and test the application easily.

As mentioned in Section 3.1.2, using the D3.JS library was a good decision that allowed to customise visualisations to fit users' requirements. However, it is considered the weakest

link in the entire architecture due to how Android communicates with JavaScript in WebView. Therefore the recommendation for future projects is to use a frameworks like React Native, as it uses JavaScript and allows to run D3.JS natively. It has the added benefit of being cross-platform.

The NoSQL database and BLOB have allowed to evolve the data structure quickly and speed up the development and testing of the processing pipeline. However, as noted in the security section 4. When using new technologies, developers should be careful. Those might have issues with vulnerabilities from untested APIs, unreasonable default settings and unclear documentation, which gives a false sense of security.

On storing JSON in client database, compression was used to save space and prevent errors being thrown. A recommendation for the future projects could be done, to also compress data on the server DB. This will save server space and reduce network requirement making the application even more lightweight.

When choosing a library for NLP processing, it was very beneficial to do early testing to verify if the library produces satisfactory results for a given task. This way, CoreNLP was chosen. Additional tests were performed when further testing the library's performance to decide between NER Tagger and NER Tagger + Coreference Resolution. Allowed to understand the performance of both parameters and choose the best one for the task, knowing exactly how they will perform. Allowed to design appropriate architecture that doesn't frustrate the users. The one problem faced with the CoreNLP library is unsatisfactory results from Location tagging and Coreference Resolution. Those can mostly be explained from the book dataset used, as the books used were classical (old), and their language can differ greatly from the models that CoreNLP might have been trained on.

The last important noteworthy technical decision was to do with where computations are performed. Having the front-end and back-end using the same language allowed to quickly move code between them to decide where this computation was to be done. In the end, designing an application that doesn't require users to have a powerful phone to get book visualisations.

Overall, architectural choices during the project sped up the development process and helped manage new features added to the application. Given that the project was delivered on time and implemented with all necessary features, the chosen architecture can be considered successful.

7.4 Results

As a result, the system was evaluated by both users and technical parameters as positive.

Users have given positive feedback based on their time with the application. They have expressed interest in the application and offered many additional modifications that they would also like to see implemented in the future. The project has achieved its technical goal of making an application accessible to people of any technical skill and doesn't need to have

powerful machinery to visualise books.

In terms of finding out if visualisation can be used as an alternative summary to literary books, the answer is not as straightforward. There are clear benefits from interviews with users that it helps them recall of the storyline and characters, but the amount of information recalled was minimal. Most likely, this is because the information presented was only confined to characters rather than combining it with anything else. Compared to (Nishihara, Ma, and Yamanishi, 2021), where their visualisation combined characters and locations have shown significant improvement in recall in their test. Therefore, visualisation presented here can improve information recall and be used as alternative summaries. However, their effect will be minimal (**Note**: no actual case study was performed, those are observations of users and my own). Combining characters with location and actions should give a lot more context to users in order for it to be an effective literary summary.

Overall, the application has satisfied all the important requirements. Technically it has been successfully implemented with robust architecture, any issues with the architecture have been highlighted and alternatives offered. Authors and Readers found this application useful but can still see more potential for this service.

8 Future Work

From Character Network evaluation 6.4, it was evident that the character network produces a useful visualisation but is limited by the current methods used for "interaction" value. Improved methods have been proposed in that section 6.3.4 that should allow getting better interaction values between characters using clustering (e.g., Kernel Density Estimates). Additionally, allowing to move processing to the user device due to the cluster producing fewer locations than the original character mentions. This reduces computation requirements in finding distances and gives users even more control of the visualisation because those values can be computed on the client device, while the clusters are done on the server.

As concluded in the evaluation, the current visualisation implementation lacks the location to be helpful as a book summary. Therefore, a new proposed visualisation method will combine character and location. As mentioned in the introduction (Nishihara, Ma, and Yamanishi, 2021) demonstrated that visualisation improves information retrieval. Their Visualisation showed characters and the location where they were. This visualisation doesn't tell us anything about character "interaction". A different visualisation produced by (Tapaswi, Bauml, and Stiefelhagen, 2014) demonstrates the visualisation of characters' appearance within a TV episode as a timeline. Combining those ideas and adding what was learned from this project (highlight what you want to see, to improve visibility). I propose the following visualisation of characters and locations (Figure 19).

The x-axis shows chapters of the book. The y-axis displays all the characters in the book. A solid horizontal bar represents the character's appearance in a chapter. Each horizontal bar is joined with a dotted line to show the character's path, allowing users to follow the characters' "footsteps".

Each chapter's characters' bars are positioned to other characters according to their "interaction" value. In other words, the vertical distance between characters represents an interaction. If characters had a lot of interaction within this chapter, then the characters would be positioned next to each other. If there was little interaction between both characters, they would be positioned as far as possible. In Figure 19, Beth and James didn't have as much interaction in the first chapter. However, in chapter two, they had a lot more interaction.

A purple dotted square is used to surround characters and show the most likely location for the group of characters. From Figure 19, we can tell in chapter one that Beth was in London, and James was in Surrey. However, in chapter two, both were in Croydon.

Too many lines and characters can make it very hard to read visualisation, especially on a mobile phone. User feedback about the highlighting function in Character Network was positive. Therefore this feature can also be used for the proposed visualisation. By tapping the character name, the user can highlight characters within the graph with different colours to highlight only what is important to the user. Both Beth and James were highlighted in this visualisation. This only highlights their paths and the locations connected to this path.

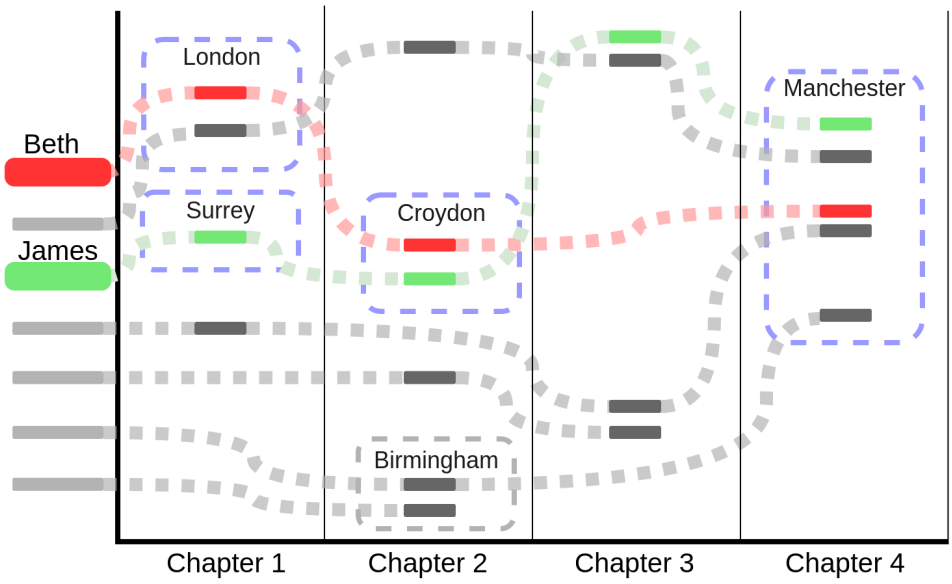


Figure 19: Character Path: Proposed visualisation, using both characters and locations extracted from literary works.

This visualisation should allow getting the characters' journey, whom they were interacting with and what places they were in. Compared to character network visualisation, this can provide an overview of the whole book rather than just a single chapter. Additionally, by only showing the closest character to one another, this visualisation solves the issue of it being too crowded like the previous visualisations. Further, because it's horizontal, it can be scrolled to see the rest of the chapters if many chapters can't fit on a phone screen.

This design was shown to readers and authors. They said this is exactly what they wanted and fits their needs perfectly. One of the author added a suggestion to allow this visualisation to be displayed in a vertical mode (by swapping x and y axis)

Location estimates can be computed by looking if there was any location mentioned within the character cluster or next to it. It is also interesting to look into how actions could be included with this visualisation (AFR3). Actions could be extracted by looking at verbs next to character mentions.

Another interesting part that wasn't done during this project was AFR1, which would allow authors to compare different versions of their books with each other. Similar to how git diff allows developers to see differences in their work. A similar project can be proposed to compare authors writing styles. From Character Network analysis, we discovered that Token and Punctuation distances were proportional. This could be because authors are consistent with their writing style, which keeps those values similar. Future projects should also look into AFR2 that can be added to entity profiles to display the scale of good to neutral to evil. The characters' actions and quotes could be analysed to determine their state. Word clouds could also be done for each character. This way, the user could get a quick overview of what the character is all about and what kind of things they say.

9 Conclusion

The project's goal has been to create a tool that allows the general public without technical skills or powerful machinery to visualise literary works. Additionally, visualisation was to be designed to give summaries for the book, allowing to quickly recall the storyline or remind of the characters present in the book.

The architecture presented here significantly lowers the entry requirements into literature book visualisation using NLP techniques. It only requires users to have the most basic phone and an internet connection to visualise any EPUB book uploaded. No prior setup is required. This architecture can further be extended or modified to work on even less powerful devices like e-readers (ex., Amazon Kindle). That could make it ubiquitous among book readers.

During user exploration of books through the tool's visualisation with the controls provided to them. The effects of the tool could be observed by helping them remember or interpret character appearances within books and finding mistakes in characters' names or surprise appearances. This demonstrates how this design could serve as a prototype for possible services to help with book reading. While the tool's visualisation implementation was only marginally useful to users, the lessons learned throughout the development allowed to make a proposition of the new methods and visualisation graph that combines both characters and location, allowing for a much deeper analysis of the characters "path" within a story.

Although, as mentioned in the literature review, plenty of research has been done into techniques that give a foundation for this project. There isn't much information regarding HCI research in literature visualisations. This is an exciting area to work in, with many visualisation ideas and information panels coming up during the project. That is an excellent area to revisit in future projects and see if there are other types of information retrieval tasks for literature that could be helpful to the target audience. Additionally, this could be extended to be a support tool for people with text comprehension issues, such as young children learning to read or people with dyslexia.

I believe that the tool presented here with further improvements described in this report can become a ubiquitous tool for authors and readers in the future. In conclusion, the architecture presented in this report has lowered entry requirements for book visualisation and presented methods for effective visualisation of character interaction and locations.

10 Glossary

Glossary

Activity A sort of the main function in Android which is specific to a particular UI Screen. 16

APK (Android Application Package) An application file ready for installation on Android device. 4, 11

Coreference Resolution When expressions refers to the same person or a thing. Example "He". https://nlp.stanford.edu/projects/coref.shtml. 21

EPUB is an e-book file format with ".epub" extension. Many e-readers support EPUB, and reading software is available for many devices. https://en.wikipedia.org/wiki/EPUB. 13

NER Named-Entity-Recognition, the process of identifying entities within a text, ex, characters, locations, organisations. 10

Token terms or words which make up documents as a sequence. 33

11 References

Neto, J.L., Freitas, A.A., and Kaestner, C.A.A., 2002. Automatic text summarization using a machine learning approach. In: G. Bittencourt and G.L. Ramalho, eds. *Advances in artificial intelligence* [Online], Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp.205–215. Available from: https://doi.org/10.1007/3-540-36127-8_20 [Accessed February 10, 2022].

Valls-Vargas, J., Ontañón, S., and Zhu, J., 2013. Toward character role assignment for natural language stories. *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment* [Online], 9(4). Number: 4, pp.101–104. Available from: https://ojs.aaai.org/index.php/AIIDE/article/view/12625 [Accessed October 6, 2021].

Not long after reading a book, i forget majority of what the book is about! is this common?, 2014 [r/books] [Online]. Available from: www.reddit.com/r/books/comments/2pll3v/not_long_after_reading_a_book_i_forget_majority/ [Accessed April 8, 2022].

Tapaswi, M., Bauml, M., and Stiefelhagen, R., 2014. StoryGraphs: visualizing character interactions as a timeline. *2014 IEEE conference on computer vision and pattern recognition* [Online]. 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Columbus, OH, USA: IEEE, pp.827–834. Available from: https://doi.org/10.1109/CVPR.2014.111 [Accessed November 8, 2021].

Brooke, J., Hammond, A., and Hirst, G., 2015. GutenTag: an NLP-driven tool for digital humanities research in the project gutenberg corpus. *Proceedings of the fourth workshop on computational linguistics for literature* [Online]. Proceedings of the Fourth Workshop on Computational Linguistics for Literature. Denver, Colorado, USA: Association for Computational Linguistics, pp.42–47. Available from: https://doi.org/10.3115/v1/W15-0705 [Accessed October 6, 2021].

Fernandez, M., Peterson, M., and Ulmer, B., 2015. Extracting social network from literature to predict antagonist and protagonist. *Recuperado de: https://nlp. stanford. edu/courses/cs224n/2015/reports/14. pdf* [Online]. Available from: https://nlp.stanford.edu/courses/cs224n/2015/reports/14.pdf [Accessed February 13, 2022].

Homakov, E., 2015. *Mongo BSON injection: ruby regexps strike again* [Online]. Available from: https://sakurity.com/blog/2015/06/04/mongo_ruby_regexp.html [Accessed April 9, 2022].

Brooke, J., Hammond, A., and Baldwin, T., 2016. Bootstrapped text-level named entity recognition for literature. *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: short papers)* [Online]. ACL 2016. Berlin, Germany: Association for Computational Linguistics, pp.344–350. Available from: https://doi.org/10.18653/v1/P16-2056 [Accessed October 7, 2021].

Matioli, L.C., Santos, S., Kleina, M., and Leite, E.A., 2018. A new algorithm for clustering based on kernel density estimation. *Journal of applied statistics* [Online], 45(2) (). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/02664763.2016.1277191, pp.347–366. Available from: https://doi.org/10.1080/02664763.2016.1277191 [Accessed April 23, 2022].

Struggling to understand stories and remember characters, 2018 [r/books] [Online]. Available from: www.reddit.com/r/books/comments/7slvbb/struggling_to_understand_stories_and_remember/ [Accessed April 8, 2022].

Jacobs, A.M., 2019. Sentiment analysis for words and fiction characters from the perspective of computational (neuro-)poetics. *Front. robot. AI* [Online], 6 (), p.53. Available from: https://doi.org/10.3389/frobt.2019.00053 [Accessed October 6, 2021].

Uday, B.K., Gogineni, K., Chitreddy, A., and Natarajan, P., 2020. Relation extraction and visualization using natural language processing. In: H.S. Saini, R. Sayal, R. Buyya, and G. Aliseri, eds. *Innovations in computer science and engineering* [Online]. Vol. 103. Series Title: Lecture Notes in Networks and Systems. Singapore: Springer Singapore, pp.633–640. Available from: https://doi.org/10.1007/978-981-15-2043-3_69 [Accessed October 6, 2021].

Nishihara, Y., Ma, J., and Yamanishi, R., 2021. A support interface for remembering events in novels by visualizing time-series information of characters and their existing places. In: S. Yamamoto and H. Mori, eds. *Human interface and the management of information. information presentation and visualization* [Online], Lecture Notes in Computer Science. Cham: Springer International Publishing, pp.76–87. Available from: https://doi.org/10.1007/978-3-030-78321-1_7 [Accessed November 10, 2021].

Pie charts in data visualization- good, bad or ugly?, 2021 [xViz] [Online]. Available from: https://xviz.com/blogs/pie-charts-good-bad-or-ugly/ [Accessed April 9, 2022].

MoSCoW method, 2022. *Wikipedia* [Online]. Page Version ID: 1082830552. Available from: https://en.wikipedia.org/w/index.php?title=MoSCoW_method&oldid=1082830552 [Accessed April 22, 2022].

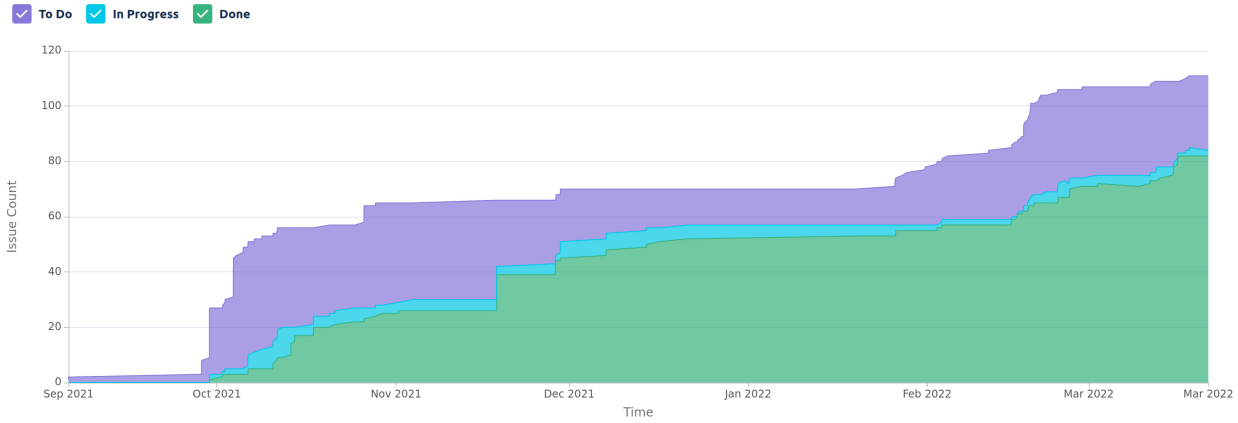
ActivityScenario, n.d. [Android Developers] [Online]. Available from: https://developer.android.com/reference/androidx/test/core/app/ActivityScenario [Accessed April 22, 2022].

ActivityTestRule, n.d. [Android Developers] [Online]. Available from: https://developer.android.com/reference/androidx/test/rule/ActivityTestRule [Accessed April 22, 2022].

Understanding memory and time usage, n.d. [CoreNLP] [Online]. Available from: https://stanfordnlp.github.io/CoreNLP/memory-time.html [Accessed April 26, 2022].

12 Appendices

A Cumulative flow diagram of my work



The graph shows the Cumulative flow diagram for my work in the past six months of the project in *Jira*.

B Commit Tags

Commit Tag	Meaning
feat:	New feature for the user, not a new feature for build script
fix:	Bugfix for the user, not a fix to a build script
refactor:	Refactoring production code, e.g. renaming variable
docs:	Changes to the documentation
test:	Adding tests, refactoring tests; no production code change
wip:	Work in progress
chore:	Updating dev scripts (e.g., CI pipeline); no production code change

C Processing Performance Testing

Book	Word Count	Laptop (m:s)	Server (m:s)
Cinderella	1441	0m:16s	0m:09s
Beauty and the Beast	5311	0m:37s	0m:40s
The Wonderful Wizard of Oz	17742	2m:54s	1m:45s
The Time Machine	18240	1m:22s	1m:49s
Alice's Adventures in Wonderland	26525	1m:1s	2m:06s
Animal Farm	30384	1m:16s	1m:49s
The Great Gatsby	48167	2m:01s	2m:49s
The Adventures of Sherlock Holmes	50648	4m:37s	4m:45s
Small Town Tales - Elliott Wright	51545	2m:4s	2m:24s
Williwaw: A Novel	52815	2m:44s	3m:51s
Untitled Drama Junior High Novel - Duncan Hui	53874	2m:02s	4m:47s
Gulliver's Travels	54651	3m:19s	4m:21s
The War of the Worlds	62479	2m:12s	3m:37s
Frankenstein; Or, The Modern Prometheus	75098	2m:53s	2m:50s
The Return of Sherlock Holmes	113873	4m:19s	5m:41s
Pride and Prejudice	121769	4m:54s	6m:49s

Table 6: Performance Testing of book processing on the server and laptop. Server has 2 GB RAM and one CPU core. Laptop has 24 GB of RAM and four Cores

D Data Ransom message

All your data is a backed up.
You must pay 0.026 BTC to 1Eft22NkrFiqDCptXAH4qUMakKBYc8ie1N 48 hours for recover it. After 48 hours expiration we will leaked and exposed all your data. In case of refusal to pay, we will contact the General Data Protection Regulation, GDPR and notify them that you store user data in an open form and is not safe. Under the rules of the law, you face a heavy fine or arrest and your base dump will be dropped from our server! You can buy bitcoin here, does not take much time to buy https://localbitcoins.com or https://buy.moonpay.io/ After paying write to me in the mail with your DB IP: recmydb+1wcct@onionmail.org and you will receive a link to download your database dump.

E BSON SQL injection protection in MongoDB

"As a client program assembles a query in MongoDB, it builds a BSON object, not a string. Thus traditional SQL injection attacks are not a problem." https://www.mongodb.com/docs/v4.0/faq/fundamentals/

F User Feedback

Feature	Authors	Readers
Entity Profile	(**Improvement**) - Want it to link from entity profile to character appearances in the text. (**Liked**) - Surprised to see one character mentioned in the chapter they thought they didn't appear in.	(**Liked**) - Different colours of characters and locations highlighted within the text help differentiate them. (**Liked**) - An entity profile is useful in getting a quick overview of where the character appeared
Entity List	(**Improvement**) - Sort entity list alphabetically rather than by mention (**Implemented**) (**Liked**) - Spotted error in the name of their character when looking through the list.	(**Liked**) - Reminded them of the characters in the book
Pie Chart	(**Liked**) - Useful when not too many characters (**didn't like**) - Hard to see less frequent characters. Authors were a lot more interested in less frequent characters. They already know who the main characters are.	(**didn't like**) - The pie chart doesn't give any useful information, and it's hard to see less frequent characters (**Improvement**) - Word Cloud visualisation could also be an option instead of the pie chart? (**partially implemented in character network**)
Lollipop Chart	(**Liked**) - A lot easier to see less and more recurring characters. (**Improvement**) - Using log scale to see lower values (**Improvement**) - Want to have control to reverse the order	(**Liked**) - Good alternative to a Pie Chart. (**didn't like**) - Because less frequent characters don't differ much from each other, they all look the same at the bottom of the graph. (**Fixed by adding mention value at the end of the lollipop**) (**Improvement**) - Use logarithmic scaling, but might lose some information.
Character Network	(**Liked**) - Controls allow them to customise their visualisation (**Liked**) - The highlight feature is fun to play with and is very useful when visualisation gets busy. (**Improvement**) - Would like the controls to be reversed to look at minor characters	(**Liked**) - Looking at the top characters is useful as this is what they are interested in. (**Liked**) - Like the controls allowing it to customise visualisation. (**Liked**) - Finds it helpful in finding the main characters within each chapter. (**Improvement**) - Would like if the app set some automatic values for the controls. (**didn't like**) - Distance method names were hard to understand their function and what they mean.
Other	- Would like to use pronouns as character references to improve the mention accuracy (Coreference Resolution) - Noted Location wasn't as accurate compared to the Character (more of the CoreNLP problem). - Overall would like to do something with the characters' emotions, relationships, and actions. It would be useful for this information to be extracted - Authors tend to know and remember the main characters. And they are a lot more interested in minor characters as this is where they can find mistakes and continuity issues.	- An excellent addition would also be to add times and periods tagging to add the context of time to visualisation. - Readers were most interested in the main characters (Most appearances) and didn't really care for the minor ones. - Find character detection more important than the location tagging, as it's more interesting to know about the characters rather than locations. - Readers found it as a useful tool for reminding character names.

G Additional UI screenshots

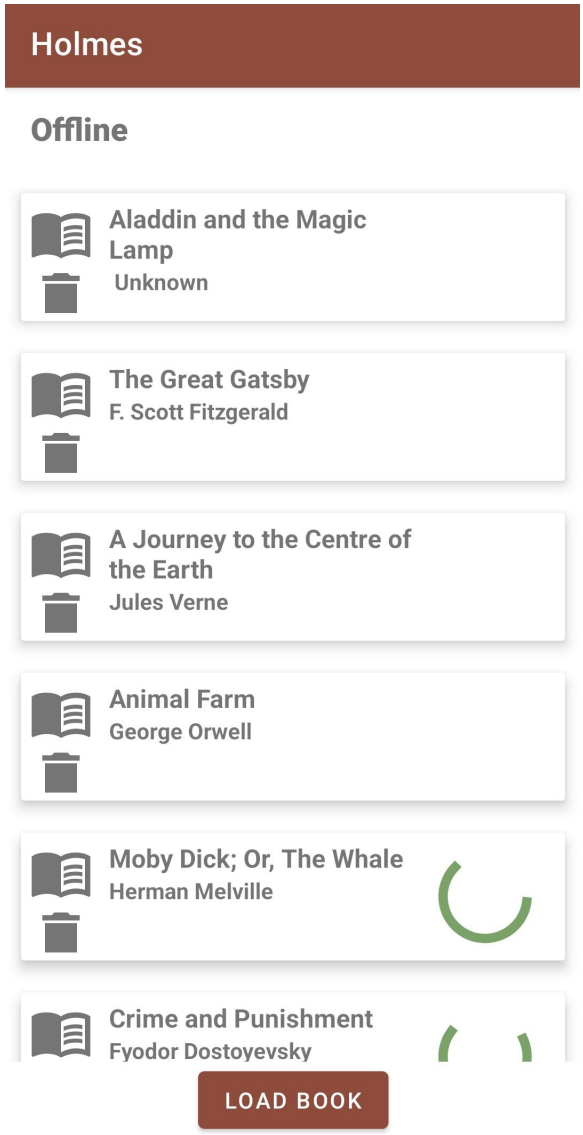


Figure 20: Main screen of the app, where user can add, delete and select a book. Also shows current connection to server and book processing status

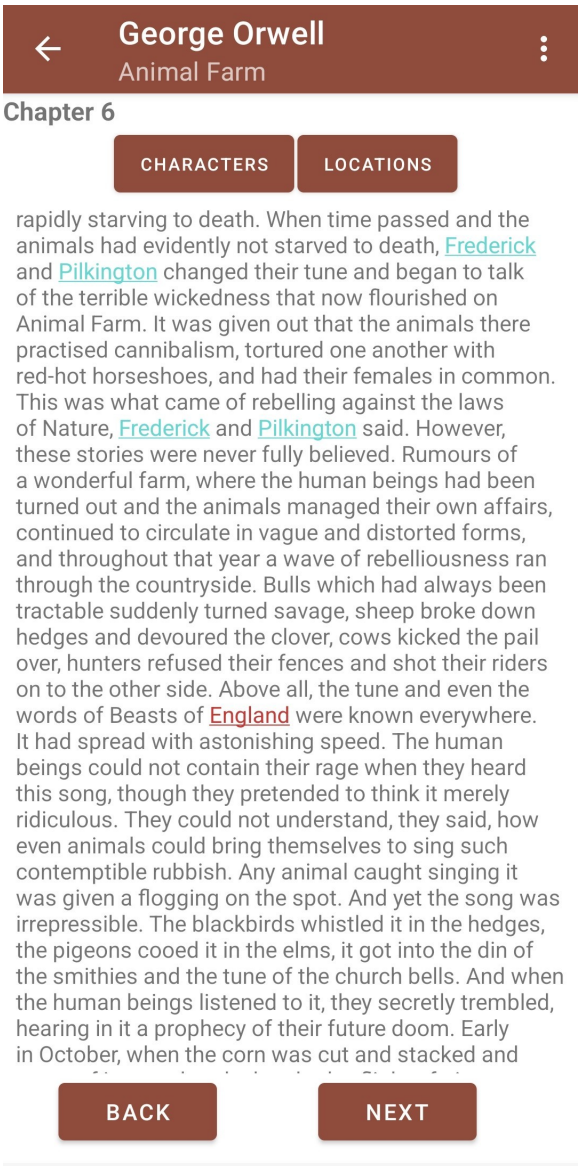


Figure 21: Reader screen, where users can scroll and switch between chapters. Characters and Locations are highlighted in the text.

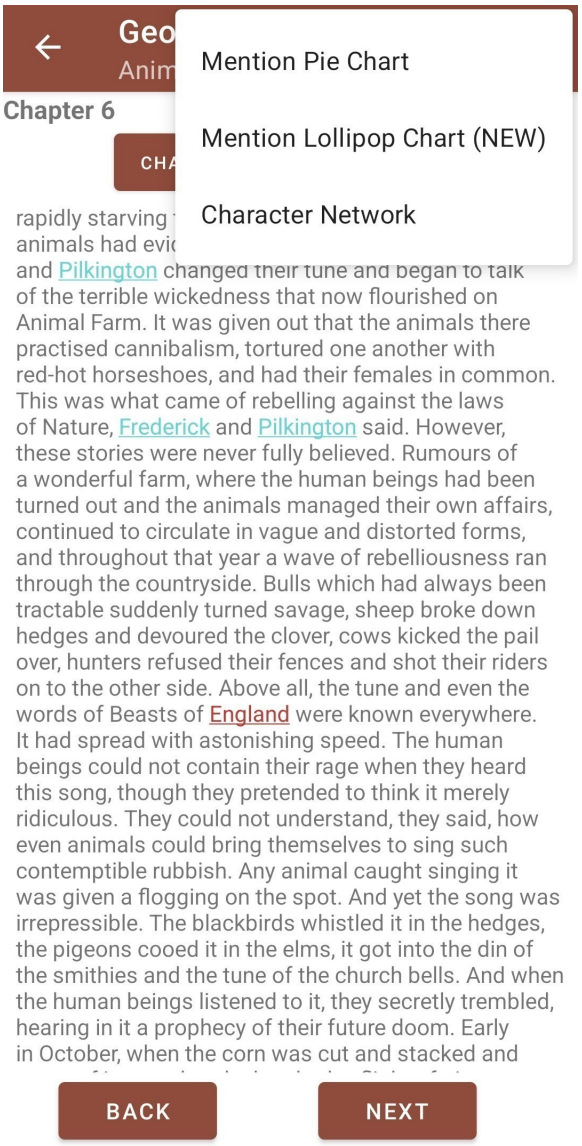


Figure 22: Choice of visualisation for a book.

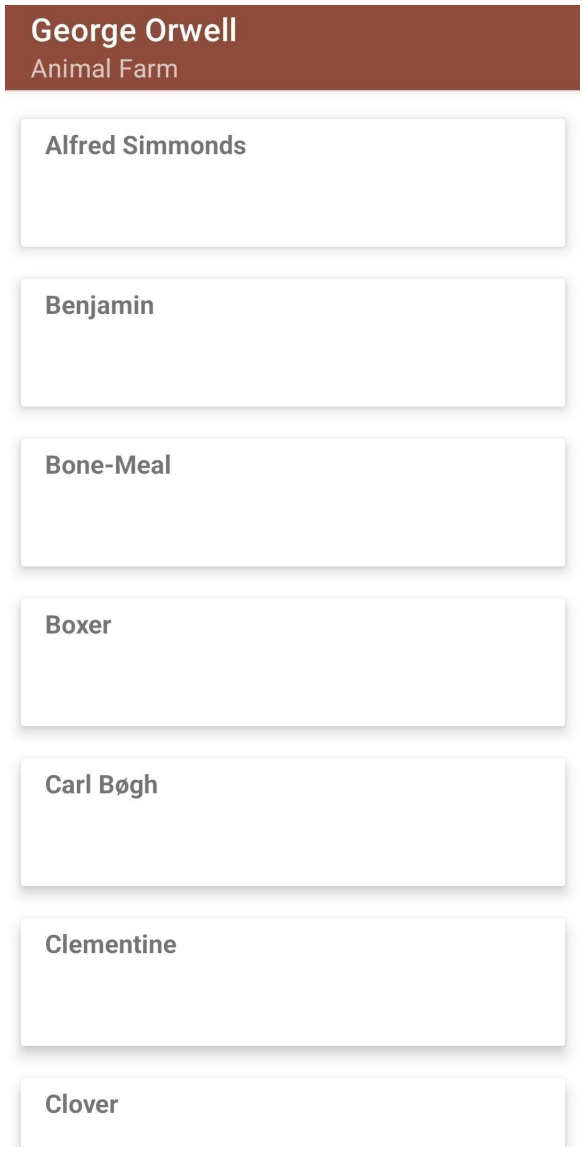


Figure 23: Lists entities for a current book in alphabetical order.

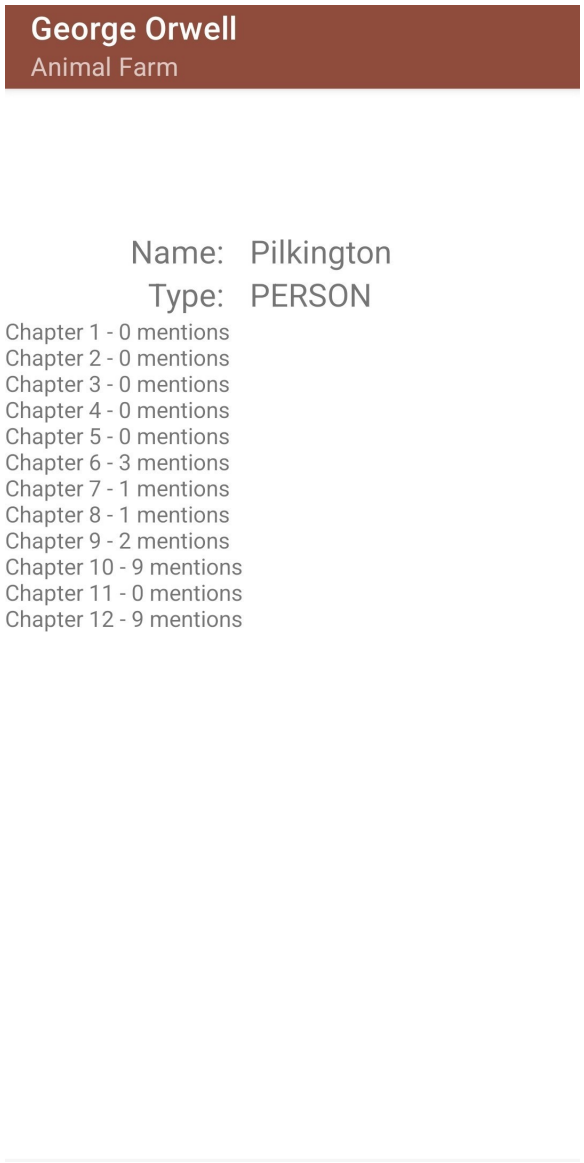


Figure 24: Entity character, displays type and number of appearances in each chapter.