Page Turner Books
Mobile Application

Group 1:

W20328872 – Brad Prosser

W20193876 – Benjamin Hogg

W2014255 – Marta Unterschute

Table of Contents

Presentation

The presentation for Page Turner Books is included in the submitted folder under the name, “Page Turner Books Presentation.” Please review that document for this section of the project.

In addition, please follow the link to visit a dummy app created to help plan and communicate the design of the application with fellow team members:

Page Turner Dummy App on Figma

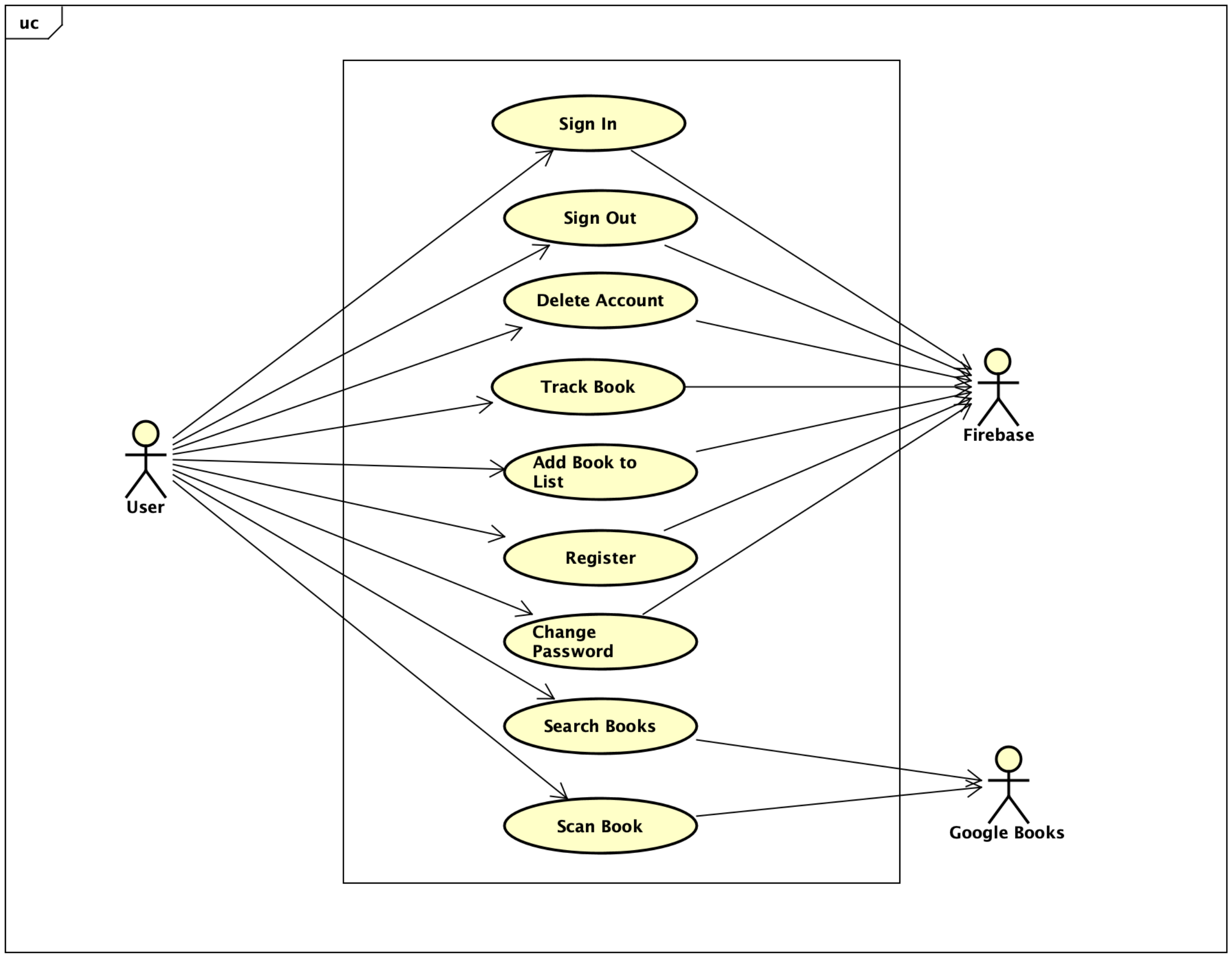
Short Report

Pre-implementation

Analysis of Design Document

After the presentation of our initial plan, we were advised to reduce the scope of the project given the complexity and time constraints. Ambitious features such as the book club which include comments, user profiles, reviews were removed from the project, partially from the quantity of work involved to develop such features but also due to the GDPR and moderation considerations of these features. Removing these aspects of the project will allow the team to focus on developing and refining the core elements of this project. A dummy app was created as a revised design document and outlines the use cases described below and reflects the changes made to the design. The dummy app illustrates the navigation, layout and functionality goals of the project except for user validation, the final UI design, how data is stored and retrieved and how security will be addressed. Considering this is still a fundamental aspect of mobile apps, these considerations are reflected in the requirement statements. The key features that were chosen for the final design were user accounts, including sign up, sign in, editing account information, retrieving books through search queries and barcode scanning utilising Google Books API, creating a database for persistent data storage, adding books to a want to read, currently reading or finished reading lists as well as a tracking book progress.

Use Case Diagram



Requirement Statements

User Input Validation

- Prevent submission of bad data
- Use warning messages or disable buttons for invalid input

Error Handling

- Catch database errors (retrieval, updating data)
- Handle navigation errors (redirect to appropriate page)
- Handle incomplete data (e.g., missing book thumbnails)

User Interface (UI)

- Adhere to Apple's Human Interface Guidelines
- Use high-contrast colors for accessibility
- Ensure clear user navigation
- Maintain consistent design throughout the app
- Apply consistent styling with specific colors and logos (branding)

User Experience (UX)

- Request permission before opening the scanner

API Data Retrieval

- Retrieve data based on ISBN and general queries
- Populate app with retrieved data

Database

- Ensure persistent data storage
- Connect data to users
- Track data related to books
- Support user account creation and deletion
- Synchronize data

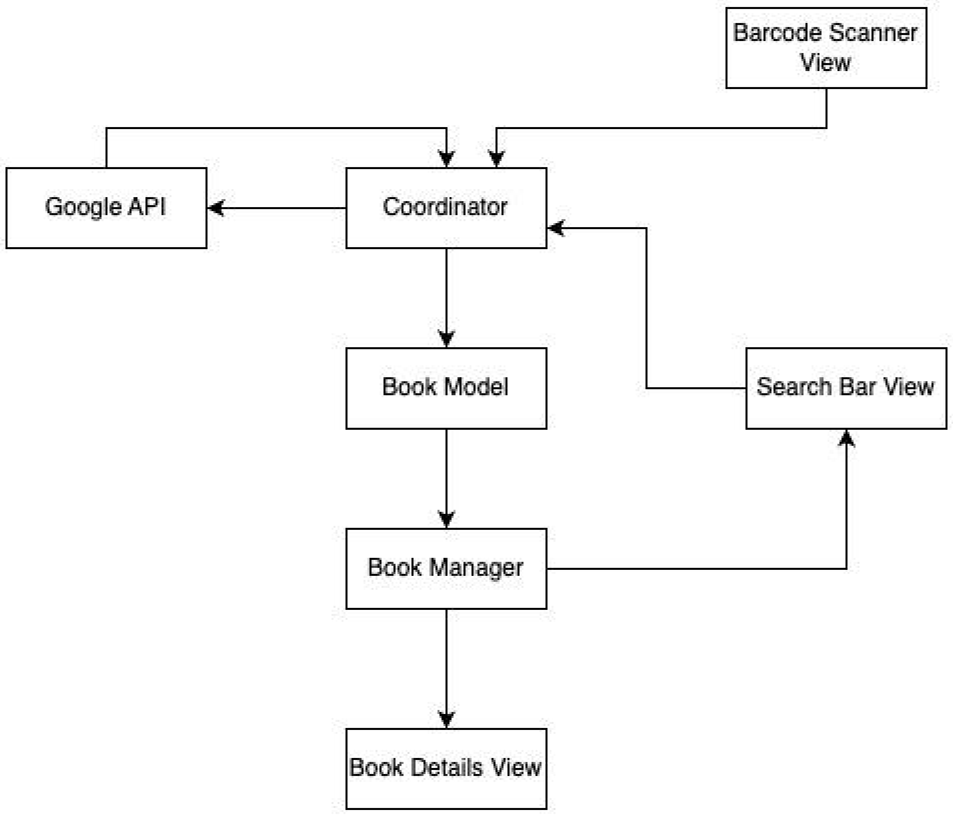
User Authentication

- Enable personal accounts with secure login credentials

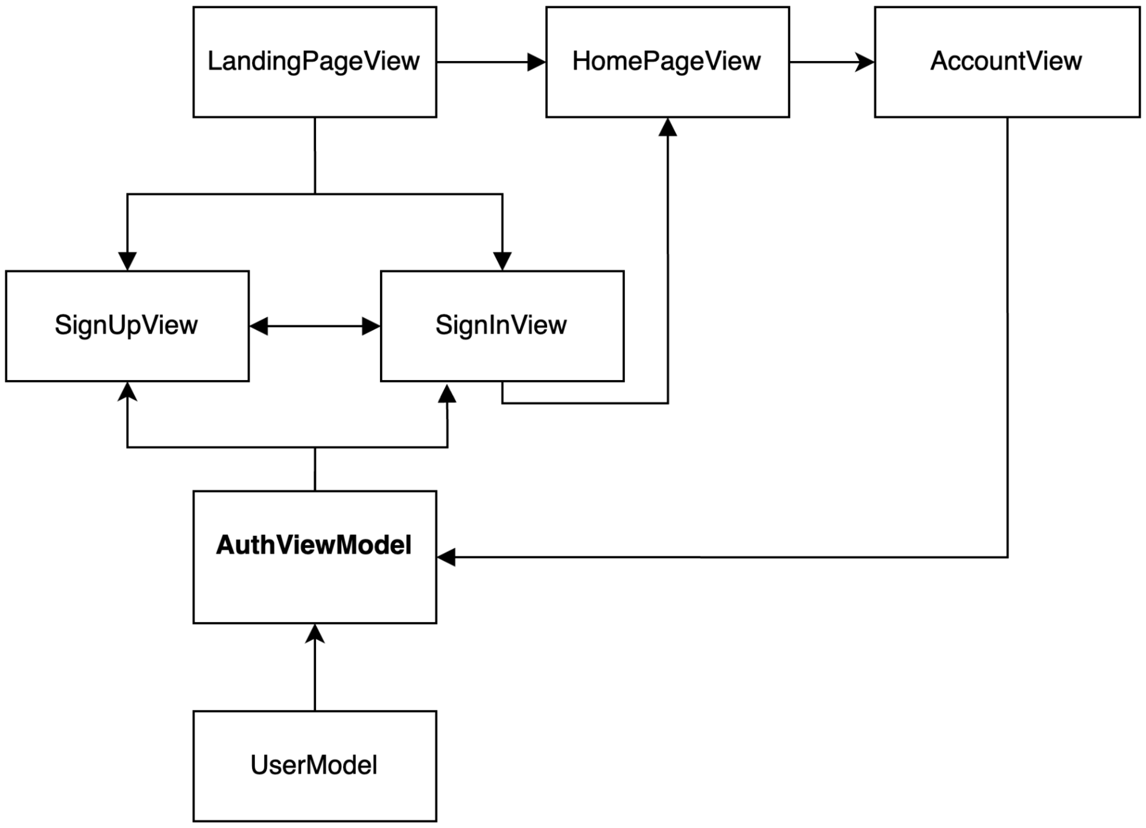
Navigation

- Provide clear navigation directions
- Use consistent components for navigation

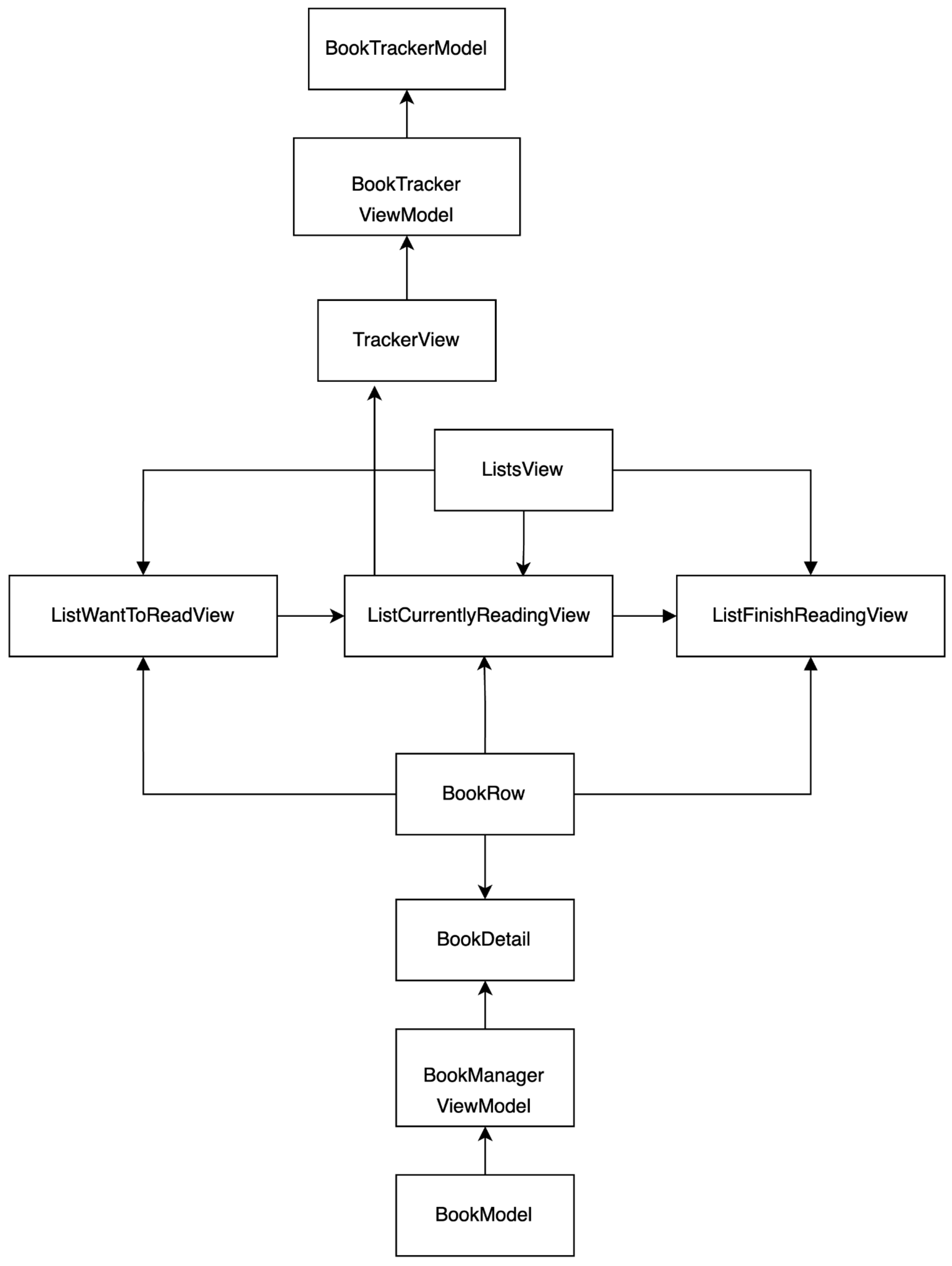
Domain Model 1: Barcode Scanner and Search Bar



Domain Model 2: Accounts



Domain Model 3: Books, Book Lists and Tracker



Post-Implementation

Critical Analysis

A critical analysis of our project shows that we achieved exactly what we set out to do - our application is fully functional both in terms of UI/layout and in terms of function and database interactions, the design and appearance is consistent across the application, and it connects successfully to the API and provides a barcode scanner. Our initial wider scope being reduced slightly following the presentation was certainly helpful, as to achieve what we have done took much more work and research than we were expecting.

The user can achieve all the required functions we initially laid out; registering, signing in, signing out, deleting their account, changing their password, searching books, scanning book barcodes, adding books to lists and tracking books. These further communicate successfully with both the Google Books API for data retrieval, and with Firebase's Firestore database for persistent data storage, data loading for users, tracking data for books and user account management, providing our application with complete functionality.

User input validation is successful across the application, with the user restricted from being able to enter erroneous data in changing of account details or in registering an account. Error handling is also successful across the application, with books unable to be added to more than one list, database errors restricted and handled where necessary, navigation errors all removed through successful use of NavigationLink, and incomplete data and loading errors being replaced with placeholder data where needed.

UI and UX has had extensive work to be as smooth and enjoyable as possible for the user; the project adheres to Apple's Human Interface Guidelines, a high contrast colour palette is implemented consistently across the application, and reusable button components are placed in similar positions in different Views to guide the user towards intuitive navigation. Design as a whole is consistent across the application through not just the colour palette and reusable components, but through similar layouts and View designs, and our branding is evidently consistent across the application.

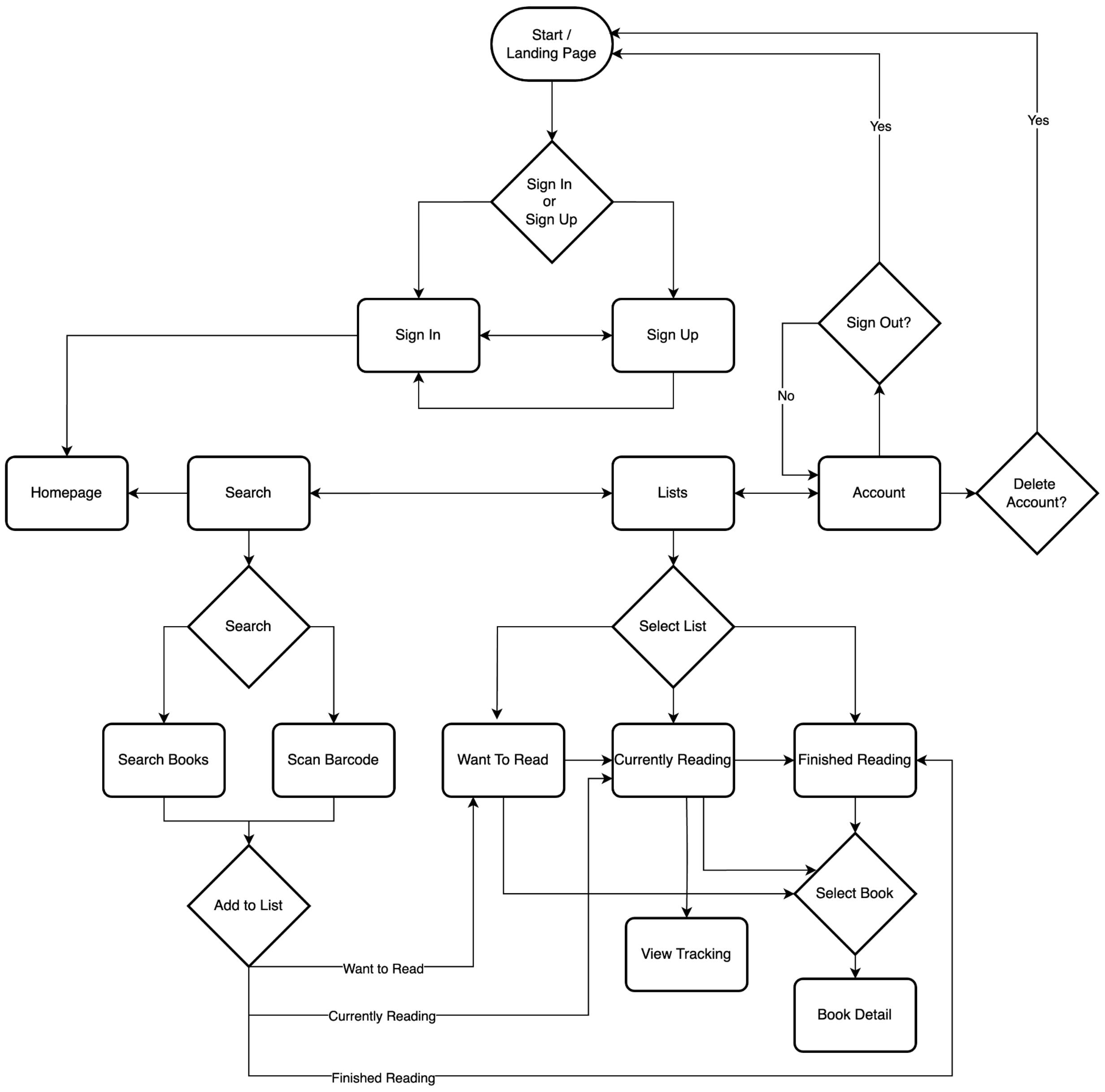
An extensive testing process through both application development and at each iterative stage of application completion ensured that all bugs and errors were quickly found, worked on and removed. Following application completion, the entire team also tested the application acting as users, confirming that all errors and bugs were fixed and the application had full functionality.

With further time to work on the project, we would have liked to introduce extra functionality and features in our application, and expand the project with the introduction of social aspects to evolve it into a book club application. Users could interact with each other in discussions about books, leave reviews on books, and have public facing profiles. In addition, some form of achievement system with badges for reading or interacting on the site could be implemented.

The project we have created lays the foundations for all of this very nicely for future expansion, though as mentioned these project evolutions were certainly far beyond achievable within the given scope and time of this project.

In conclusion, the hard work and dedication of the team has created an application with full function and consistent and appealing UI and design. With this experience behind us we could no doubt produce a similar application more quickly and efficiently, but the project served as an invaluable learning journey, and we are proud to have produced an application that achieves the goals we set out and to see the progression from a conceptual dummy application, to a fully functioning iOS application.

User Flow


Start / Landing Page
Sign In or Sign Up
Sign In
Sign Up
Yes
Yes
Sign Out?
No
Homepage
Search
Lists
Account
Delete Account?
Search
Select List
Search Books
Scan Barcode
Want To Read
Currently Reading
Finished Reading
Add to List
Select Book
View Tracking
Book Detail
Want to Read
Currently Reading
Finished Reading

Application Testing

Test #	Title	Expected Outcome	Pass / Fail
User Account Tests			
1	Create User	Upon successful user registration through the application, a document for a userId is created in the Users collection in Firestore database.	Pass
2	Sign In	A registered user can successfully sign-in to the application using their email and password.	Pass
3	Sign Out	A logged in user clicks log out and upon confirmation is redirected to the landing page with no access to the application without signing in again.	Pass
4	Change Password	A logged in user can change their password, and when they try to login with their old password their log-in will fail until they use the new password.	Pass
5	Delete Account	A logged in user clicks delete account and upon confirmation their userId document is deleted from the Users collection in Firestore database and the user is redirected to the landing page with no access to the application without registering again.	Pass
Barcode Scanner & Google Books API Tests			
1	Barcode Scan	A logged in user can open the barcode scanner, scan a valid barcode, and the correct barcode will be scanned and logged.	Pass
2	Barcode to ISBN Conversion	A logged in user can open the barcode scanner, scan a valid barcode, and the related valid ISBN will be retrieved.	Pass
3	ISBN Query to Google Books API	A logged in user can open the barcode scanner, scan a valid barcode, the related ISBN will be retrieved, and the Google Books API will be queried and return the correct book details to the user.	Pass
4	Text Search Query to Google Books API	A logged in user can open the text search menu, enter valid details for a book, and the Google Books API will be successfully queried and return the correct book/s.	Pass

User Books & Lists Tests			
1	Add Book to ‘Want to Read’ List	When a user clicks to add a book to the Want to Read list in the application, the wantToRead collection will appear (or be appended to) in Firestore for the user’s userId document, and it will contain the bookId document. The book will also appear in the list in the application for the user.	Pass
2	Add Book to ‘Currently Reading’ List	When a user clicks to add a book to the Currently Reading list in the application, the currentlyReading collection will appear (or be appended to) in Firestore for the user’s userId document, and it will contain the bookId document. Within this document will be a tracking collection, which will contain a trackingData document for that user’s book. The book will also appear in the list in the application for the user.	Pass
3	Add Book to ‘Finished Reading’ List	When a user clicks to add a book to the Finished Reading list in the application, the finishedReading collection will appear (or be appended to) in Firestore for the user’s userId document, and it will contain the bookId document. Within this document will be a tracking collection, which will contain a trackingData document for that user’s book. The book will also appear in the list in the application for the user.	Pass
4	Delete Book from ‘Want to Read’ List	When a user clicks on the trash can icon to delete a book, a confirmation pop-up will appear, and following confirmation here the related bookId document will be deleted from the user’s wantToRead list in the Firestore database. The book will also no longer appear in the list in the application for the user.	Pass
5	Delete Book from ‘Currently Reading’ List	When a user clicks on the trash can icon to delete a book, a confirmation pop-up will appear, and following confirmation here the related bookId document (and its nested tracking collection) will be deleted from the user’s currentlyReading list in the Firestore database. The book will also no longer appear in the list in the application for the user.	Pass
6	Delete Book from ‘Finished Reading’ List	When a user clicks on the trash can icon to delete a book, a confirmation pop-up will appear, and following confirmation here the related bookId document (and its nested tracking collection) will be deleted from the user’s finishedReading list in the Firestore database. The book will also no longer appear in the list in the application for the user.	Pass
7	Move Book from ‘Want to Read’ to ‘Currently Reading’ List	When a user clicks on the ‘Start Reading’ button next to a book in their Want to Read list, a confirmation pop-up will appear, and following confirmation here the related bookId	Pass

		document will be copied to the currentlyReading list for that user in the Firestore database, have a tracking collection and trackingData document created for it, and be deleted from the user’s wantToRead collection in the database. These changes will also be successfully reflected in the application.	
8	Move Book from ‘Currently Reading’ to ‘Finished Reading’ List	When a user clicks on the ‘View Tracking’ button next to a book in their Currently Reading list they will be taken through to the TrackerView, where they can click ‘Finish Reading’. A confirmation pop-up will appear, and following confirmation here the related bookId document will be copied to the finishedReading list for that user in the Firestore database, have its tracking collection and trackingData document moved with it, an endDate added, and be deleted from the user’s currentlyReading collection in the database. These changes will also be successfully reflected in the application.	Pass
9	View Tracking	When a user clicks on the ‘View Tracking’ button next to a book in their Currently Reading list, they will be taken through to the TrackerView, where they can view their saved tracking information related to that book (thanks to the nested tracking collection containing the trackingData document in the Firestore database).	Pass
10	Can’t Update Last Page Read Beyond Total Pages	When a user selects edit to enter in the last page they read in their book to update their progress, if they enter a page number beyond the total number of pages in the book then it will automatically save the total number of pages so their progress does not exceed 100%.	Pass
11	Edit Tracking - Page Progress	When a user edits the Page Number in TrackerView, the progress bar updates, and the updated number is saved in the trackingData document in the Firestore database.	Pass
12	Edit Tracking - Start Date	When a user edits the Start Date with the date picker in TrackerView, this information is updated and saved in the trackingData document in the Firestore database.	Pass
13	Edit Tracking - End Date	When a user views a book in their Finished Reading list and edits the End Date with the date picker, this information is updated and saved in the trackingData document in the Firestore database.	Pass
14	Tick Mark in Add to List Dropdown Menu	When a user clicks the ‘Add to List’ button in Book Details, if the book already exists on one of their lists then a tick mark is displayed next to the list containing the book.	Pass
15	Addition of Book to Multiple Lists Prevented	If a user tries to add a book to more than one list at a time, they receive a pop-up alert informing them it is already added to one of their lists.	Pass

Ben’s Contribution Summary

Throughout the project, my specific contributions were important in developing critical components such as the barcode scanner, search bar, book model, and the integration of the Google Books API. These contributions significantly enhanced the functionality and user experience of our application.

I began by looking into the Google Books API to understand its capabilities and determine how best to integrate its features into our application. This involved a review of the API documentation, focusing on essential aspects of the book model that we could use. I identified key data points, such as title, author, publisher, and ISBN, that would be crucial for our application.

To better understand the Google Books database, I utilized RapidAPI, which provided a user-friendly interface for testing and reviewing JSON responses. This gave me a clear understanding of the data structure and helped me to replicate the necessary fields in our application. I then planned the use of various query options, ensuring we could handle a wide range of searches, from broad keyword searches to specific ISBN queries.

The next phase involved implementing the barcode scanner. I adapted a barcode scanner library based on several online guides and resources, ensuring compatibility with our application's requirements. This process included configuring the scanner to work on mobile devices, allowing users to scan book barcodes using their smartphones, requesting permission from the user, and bypassing Swift’s HTTPS restrictions.

Once the scanner was operational, I developed methods to parse the ISBN from the scanned barcode. This involved writing code to extract and validate the ISBN, which was then used to initiate a query to the Google Books API. The successful retrieval of book information from the API marked a significant milestone in the project.

To efficiently manage the data flow from the barcode scanner, I implemented a coordinator. This coordinator was responsible for parsing the ISBN, sending the query to the Google Books API, retrieving the JSON response, and constructing a book object within our application. This comprehensive approach ensured smooth data handling and integration.

Building on the barcode scanner's functionality, I developed a search bar feature to allow users to search for books using keywords or titles. This required creating a general query system for the Google Books API, which could handle both specific and broad search terms. I enhanced the coordinator to differentiate between data originating from the barcode scanner.

I expanded the book model to support a collection of books, rather than a single book entity. This modification was crucial for accommodating multiple search results and providing users with a view of all relevant books. To optimize component reuse and streamline data management, I introduced a Book Manager ViewModel. This ViewModel was tasked with managing and storing book objects, facilitating efficient data handling and retrieval.

The development of the API search through barcodes or general queries and their associated views, as well as the modeling of the book object, laid a solid foundation for the future development and integration of login and tracking features that my colleagues later developed.

Beyond my specific technical contributions, I actively helped with my team in brainstorming sessions, design discussions, and project coordination. I played a role in documenting our progress and communicating amongst team members. As my colleagues developed login and tracking functionalities, I continued to troubleshoot integration issues related to API searches and barcode scanning, ensuring a cohesive and functional application. Ultimately, I think the app turned out great and I’m really proud of the group's efforts.

Brad’s Contribution Summary

My project contributions began by creating the Firestore Firebase Database for persistent storage for our application. This involved connecting the Firestore database to our Xcode and SwiftUI project, learning how NoSQL databases function, and learning how to perform CRUD operations on NoSQL databases through our SwiftUI application.

I created the AuthViewModel to handle all user based functions, such as the expected signIn, signOut and createUser. I then built on this ViewModel further to introduce an updatePassword function for the user, which involved an additional necessary validateCurrentPassword function to adhere to the Firestore Database rules. I further extended the functionality of the AuthViewModel with a deleteUser function for the user to remove their account and data; this latter part was especially difficult, as within the database I had set up nested collections of books lists and tracking data within each user collection, so deleting a user required careful deleting of each sub collection to complete fully.

These nested collections were my next contribution to the project, as I set up book lists (currentlyReading, wantToRead, finishedReading) that would be created when a user adds a book to a list as a sub-collection within the userId document in the database. The books in the sub-collection lists are then stored as documents under bookId, and within each book document for a user in a list is a further sub-collection called tracking, which then holds a trackingData document of that book for that specific user. Multiple documents at a higher level (e.g. Users, Books, Tracking) would have been my preferred database setup, but seeing as this seemed more related to relational databases, I changed my approach to utilise the nested collections that are apparently more suited to a NoSQL database like Firestore.

The logic for CRUD operations to these sub-collections are handled in SwiftUI by the BookListsViewModel and the BookTrackerViewModel. This was the most difficult part of the project for me, as I wanted the two ViewModels to perform a multitude of tasks beyond basic operations, and as such the number and complexity of their functions increased as the project progressed. Basic functions such as loadBooksFor, addBookToFirestore and deleteBookFromFirestore were fairly simple, but when I wanted custom behaviours such as moving books from one list to another, or adding a book to a list but not creating tracking data if it was on the wantToRead list, things became more complex.

For example, a user completing a book in the TrackerView required the book details to be copied from the currentlyReading to finishedReading list (while also not falling foul of my function preventing a user from adding a book to two different lists), deleting the trackingData from currentlyReading in the database, and then updating the trackingData in finishedReading to display the end date. Other complex functions included moveBookToCurrentlyReading (copy details from wantToRead to currentlyReading, add tracking data, delete data from wantToRead list), and functions within the BookListsViewModel to update the endDate and the startDate for book tracking dependent on user customisation or a book being moved between lists.

My focus was first on finishing successful setup of the logic of these functions for our program, and making sure they also interacted properly with Ben's excellent barcode scanner and book search functionalities. As such, the UI I initially created to test the application behaviour was very crude, and thankfully I had the opportunity to improve on this by joining Marta in working on some of the UI and appearance aspects of the project towards the end. She had already worked extensively on these when I joined this stage of the project, so my task became assisting in some design decisions and moulding various Views to match the very aesthetic UI and design themes and choices she had made across the program.

I implemented user validation messages across the Views, ensuring error messages in the AccountView guided users towards required inputs in changing password, and providing warnings to confirm sign out and account deletion. I also implemented confirmation pop-ups each time a user wished to delete a book from a list, or move a book between lists. The flow of UX through the Lists was also something I had in mind when building the ViewModels, and I then expanded upon this while working on the UI; the idea being a user can add a book directly to any list, but that a book can only exist on one list at a time. They can move a book from Want To Read to Currently Reading by clicking start reading, which then implements tracking that they can view in the Currently Reading list (and homepage) through view tracking, where they can find options to enter page number progress, edit the start date, and also finish reading - which will move the book to the Finished Reading list, with a customisable end date.

Delving deeper into pure UI and appearance, I assisted in ensuring navigation throughout the views was implemented - the most frustrating bug of the project for me coming through one NavigationLink leading to another NavigationLink in two views, resulting in two pages loading on one click. I was surprised at just how much time proper design and UI take to achieve, and have certainly gained a new appreciation for the work put into these areas in iOS applications.

Other UI work involved adding image backgrounds for the BookSearchView and the ListsView, helping to apply the chosen colour palette theme across the views, organising page layouts and appearances such as the BookDetailsComponent, and spending a lot of time fixing numerous small bugs and errors that would arise from clashing UI components in one view, or data passing between different views.

It has been a very rewarding experience to see an application concept grow from thought, to mockup design, to a fully functioning iOS application, and I'm lucky to have worked with two incredibly capable programmers on this project who shared in the drive and desire to achieve all of our project goals.

Marta’s Contribution Summary

At the project's onset, I led discussions regarding the features and functionality of our website, introducing Figma to the team. I helped the team use this software for our storyboard creation, refining designs for our presentation. Following the astute recommendation to narrow the scope of our project, I analysed the remaining features, their interactions and the navigation across the app. I then created a dummy app to clarify the structure and flow of our product which aided in communication and understanding of our goal. I led the discussion for our small report, adding considerable input for the analysis, system requirements, use case scenarios and finalised the analysis, two of the domain models and the flow diagram for our report.

With our new minimal viable product, our roles became more specialised due to these reduced features. Brad assumed the role of database developer using Firebase as well as connecting the logic between the view models and our database. Ben managed our API connection with Google Books API and integrated it with searching the books through text queries and barcode scanning as well as making the base components for Book Detail and Book Row. My responsibilities shifted towards the front-end development as well as the functionality of the Book Tracker while also creating the base model and view model which Brad completed in order to connect it with the backend.

I researched Apple’s Human Interface Guidelines to ensure our design was professional and emphasised consistency and clarity across the components and user interactions. This included, but not limited to, focused states, disabling/enabling buttons, confirmation messages, placement, spacing, alignment and spacing. I collaborated with Brad regarding the navigation. I created the base navigation between pages and navigation stacks, including the navbar and some transitions between pages, Brad added or adjusted navigations when additional database interactions were included as he developed them. We worked in tandem testing these navigations were functioning as expected across the views.

The design was an iterative process and involved multiple redesigns and implementations to get our features functioning correctly with the backend and the UI while also adhering to Apple’s guidelines and any technological limitations. To aid in this extensive process, I built many reusable components, such as a variety of buttons, custom modifiers for button styling, text fields, flipping images, and menu styles, to ensure consistency throughout the app as well as to make faster changes during this process. Some of these components made it to the end of the design process and some were ultimately removed.

In the end, I crafted the UI for the Homepage, Sign Up, Sign In, Tracker View, Navbar, the Landing page, including adding animations to the logo, as well as joint efforts with Brad for the Book Detail component, Account, Currently Reading, Finished Reading, and Wanting to Read List as well as integrating the Book Rows. I oversaw any additional UI changes contributed by my team members to ensure they aligned with the design principles. I added mock data and previews across the program to aid in development. Collaborating with Brad, we thoroughly tested the program and functionality, and put in considerable work debugging UI and UX errors, ensuring a smooth user experience.

Working with my team members was a delight. Everyone was hardworking and communicated well as we developed our mobile application. It was truly a team effort.