CFG Project Assignment: Bound 📚🪄

Introduction

The goal of this project is to collaboratively enhance both our team-building and coding abilities by developing a web-based application.

Bound leverages a public API to search through thousands of books online, combining the unique literary preferences of two users. Many book club members, friends, and families face the challenge of finding a book they both enjoy due to the vast array of genres, authors, and writing styles available.

Our application stands out as the first of its kind to blend user interests into a tailored book recommendation, driven by our shared love for reading.

This project will guide you through the following sections:

- **Background:** An exploration of competitor apps, and what sets our app apart from existing products on the market.
- **Specifications & Design:** An overview of the app's design, architecture, and the key specifications required to achieve our Minimum Viable Product (MVP).
- **Execution & Implementation:** A breakdown of team responsibilities, task implementation processes, and the tools used to bring the app to life.
- **Testing & Evaluation:** Insights into our testing methodologies, the limitations we encountered, and potential improvements we identified.
- **Conclusion:** Final reflections on the project, key takeaways, and an assessment of whether we met our initial objectives.

Background

Since the pandemic, there has been a surge in interest in reading, with US print book sales rising by 8.9% in 2023[1]. Book clubs have also experienced significant growth, with younger readers increasingly viewing reading as a social activity. Eventbrite, for example, reported a 41% increase in book club listings between 2022 and 2023[2].

Several apps currently serve readers by helping them discover books they might enjoy. Leading platforms like Fable and Goodreads focus on fostering community-driven reading experiences. Other niche apps, such as StoryGraph, provide personalized features like mood-based recommendations and detailed customization options.

---

[1] https://www.tonerbuzz.com/blog/book-and-reading-statistics/

[2] https://www.theguardian.com/books/2024/feb/29/uk-in-the-midst-of-a-boom-in-book-clubs-as-gen-zs-hobbies-change

However, no existing app specifically combines the shared interests of two users to recommend a book they would both enjoy. **Bound** addresses this gap with a streamlined user flow:

Bound is designed to make discovering shared literary interests both fun and effortless, catering to a growing audience of socially-minded readers.

Specifications & Design

Our team began by compiling a detailed list of specifications necessary for the application. From the outset, we categorized features into "must-haves" and "desirable" to prioritize tasks effectively. This approach proved particularly valuable as our group size fluctuated over time. Below is the prioritized list of features:

Specifications

Must-haves	Desirable

• Onboarding & Auth
  ○ User log-in (sign-in, sign-up, forgotten password)
  ○ User library including saved books
  ○ Add friends functionality
	• Onboarding & Auth
	  ○ Profile set up with profile picture, location, currently reading
	  ○ User library including saved books

• Book Discovery
  ○ Ability for user to connect in a "bind" with a friend
  ○ User recommendation based on bind result
  ○ Ability for a user to purchase a book via an affiliate link: https://uk.bookshop.org/
  ○ Users can see the image of their "bind" result with descriptions for accessibility
	• Book Discovery
	  ○ Invite link to send to a friend to join the "bind"
	  ○ User library filters eg. "read", "currently reading", and "wishlist"
	  ○ User library including "bind" result books

	• Other
	  ○ Add friends functionality
	  ○ User library filters eg. "read", "currently reading", and "wishlist"
	  ○ Accessibility: ability to zoom into the app and change font sizes.
	  ○ User book club page set up
	  ○ User book club invite functionality

Design

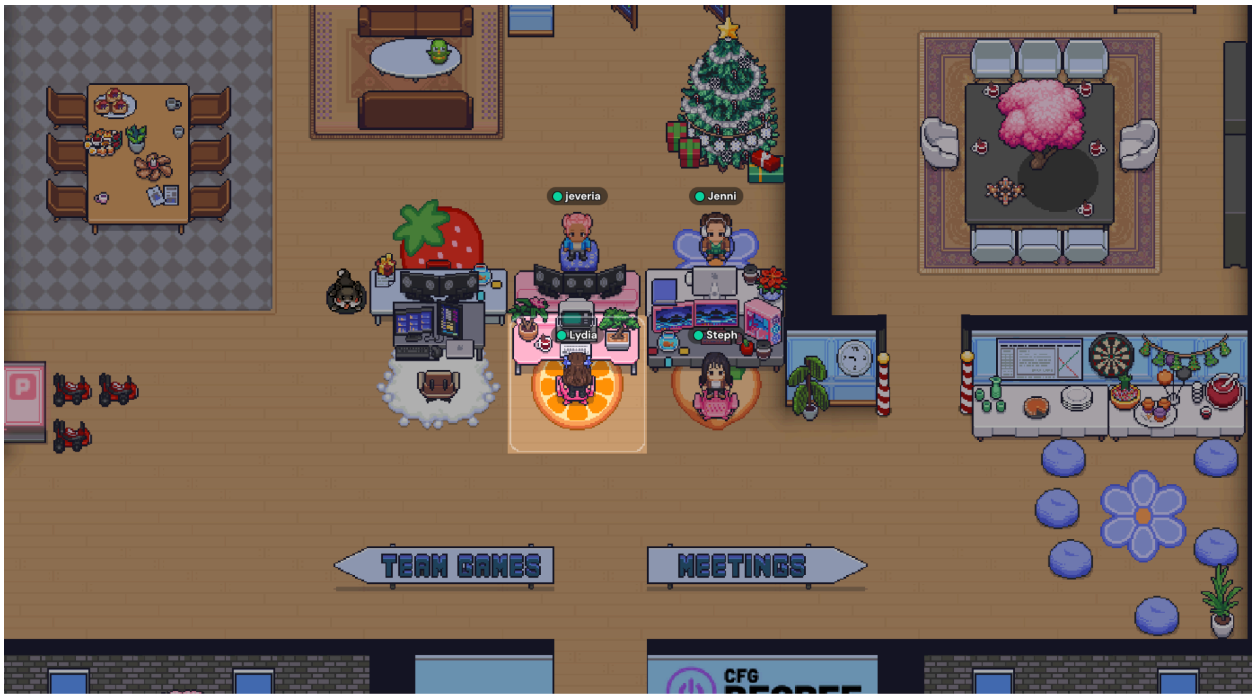
We began by using Figma to create high-fidelity UX/UI wireframes for the app.

The initial wireframes were developed in a mobile format and included both our "must-have" and "desirable" features. This process allowed us to visualize the number of components required for various sections of the site and identify opportunities for reusing and repeating code effectively.

Our design was heavily inspired by the aesthetic of dark academia. To ensure accessibility, we opted for high-contrast visuals, using light text on dark backgrounds. In the "Design Playground," we curated references for colors, fonts, and logos to maintain consistency throughout the design process.

We then refined our focus, selecting the specific pages required for the MVP product, and created high-fidelity wireframes in a desktop format. To enhance usability, we designed the layouts in a "Z shape," making the content intuitive and easy to scan.

Implementation & Execution



From the outset of the project, we used Gather as our virtual shared workspace for regular meetings, collaboration, and idea-sharing. This platform proved invaluable for screen sharing during pair programming sessions, and frequent check-ins helped keep us aligned with our deadlines.

As a team, we initially divided tasks into **frontend**, **backend**, and **API** groups. However, when two team members departed, we adapted by leveraging the agile methodology, allowing us to reorganize

responsibilities effectively. To maintain transparency and track our progress, we created an Activity Log to document completed tasks and monitor the hours spent on the project.

Execution Phases

1. GitHub

We began by setting up a GitHub repository to manage our codebase. Using Visual Studio Code, we remotely connected to the repository and started building our React app. To streamline development, we installed essential dependencies, including **Vite**, which helped create a fast and efficient development environment.

With **Node.js**, we established the foundational structure for our React desktop app. We then created the required pages for our Minimum Viable Product (MVP), such as **authPages**, **bind**, and **homepage**, and ensured their URLs were properly imported into the **App.js** file for seamless navigation.

At this stage, we made a team decision to consolidate all our CSS code into a single universal stylesheet, **index.css**. Class names were carefully chosen to clearly reflect their purpose and relation to the site, keeping the codebase organized and maintainable. The **index.css** file was methodically structured to break down styling for individual sections of the pages, making it easy to navigate and edit specific areas when needed.

2. Frontend

The frontend of our site was originally going to be managed by Lydia and Emma, however, this changed when Emma left the project. Lydia wrote the CSS code for each of the pages, and Lydia, Jeveria, and Jenni, and Beth jointly wrote parts of the JSX code.

a. Process

We started by writing the JSX code to create the necessary structure of the site. This provided the foundation for our pages, mirroring the design and functionality outlined in our initial wireframes.

Next, we applied CSS styling to the site, closely referencing the design decisions made in Figma.

One key learning experience came with using React's **useState** hook. On a specific page, we implemented a function that dynamically updated the content based on the user's actions. For instance, when a user's bind was completed, the **useState** hook changed its value from **false** to **true**, triggering the display of the bind result over the initial page.

function BindPage() {

```
 const [isBindComplete, setIsBindComplete] = useState(false);
 //Additional code for functionality
}
```

b. Components

We developed several core components to enhance the functionality and consistency of our site, such as a universal site footer and an image carousel. These components were designed to be reusable, streamlining the development process by reducing redundancy. Key components performed essential functions, including fetching and displaying book images as well as retrieving book ratings, ensuring a seamless user experience across the platform.

c. Tools

For the frontend, we utilized React.js with Vite, which enabled rapid and efficient development. To design the user interface, we relied on Figma for creating and refining all design files. Accessibility was a priority throughout the project, so we incorporated tools like Adobe Color to verify that our color schemes met WCAG compliance standards for adequate color contrast.

3. Backend

Jeveria and Jenni managed the backend of the project, and focused on connecting the APIs to the frontend code. We used fetch and axios to link the frontend and the backend and connected the function to buttons. This calls the API when the corresponding button is clicked by the user.

a. Process

We set up the Node.js server to handle API calls and manage user authentication. We implemented the database schema to store user profiles, bind results, and book preferences. We also integrated public APIs, such as Google Books and NYT Books, to dynamically fetch book data. To ensure secure operations, We designed RESTful endpoints for user authentication (using tokens or session-based auth), book searches and recommendations, and data storage and retrieval of user preferences.

b. Tools

Node.js with Express was used for backend logic, while SQL was utilized for relational data storage. Libraries such as bcrypt were implemented for password hashing, and JWT was used for authentication token handling.

4. API/SQL

Beth and Steph were responsible for overseeing the API integration and managing the SQL database functionality of the site. Bound.db the database was created with various tables including members, profiles and favourite books with the appropriately linked foreign keys.

a. Process

We fetched data from trusted sources like the Google Books API for reliable book information and platforms like Bookshop.org for affiliate monetization.

Middleware has been implemented to optimize API calls and data parsing for efficient friend searching and book binding. We used robust API endpoints to manage user data, book reviews, and favorite lists, while the Google Books API enables seamless book search functionality.

Our features like search-as-you-type and SQL-powered friend search enhanced the user experience, allowing users to quickly find books and connect with friends. Users can add books to their favorite lists and wishlists, with error-handling logic providing clear feedback for any issues. Sensitive data, such as API keys and database credentials, is securely managed using a **.env** file, ensuring confidentiality. Close collaboration with the frontend team ensured APIs meet data requirements and provided a smooth user experience.

b. Tools

The backend of the project used Node.js for the server-side APIs, including Google Books' public API. SQL was used throughout the backend of the project, to store data about books retrieved from the Google Books API, and member data which was plugged into the sign-up and log-in components.

c. Security

Security was a fundamental consideration throughout the development of Bound to ensure the protection of user data and readability of the application. Only secure and trusted APIs were integrated, with additional validation for API responses.

Challenges & Solutions

One of the most significant challenges we faced was the reduction in team size. We began the project with seven members but completed it with five. This shift increased individual workloads, with some team members taking responsibility for entire sections of the site.

We addressed this challenge through regular check-ins and an iterative approach. By stepping outside of our assigned "roles," we collaborated to tackle different areas of the site when someone encountered difficulties. The use of **Gather** facilitated pair programming, allowing us to focus on and resolve specific coding challenges together.

Testing & Evaluation

Unit Testing played a crucial role in verifying the functionality of individual modular components in isolation. By using dummy data, we were able to test the logic of functions without relying on external dependencies like databases or APIs. For example, on the homepage, unit testing ensured that books and friends consistently displayed reliable and predictable outcomes, reinforcing the robustness of the codebase. For the first two pages, Beth tested through the browser, Postman, and by inspecting the console. Lydia regularly inspected the console for CSS purposes.

Unfortunately, there are limitations to the app due to time constraints. Currently, the sign up flow and the favourite books pages work. However – while components of the homepage work with dummy data – they are not fully connected to the API. The "bind" feature is very close to working, and the pages currently work by manually changing the "useState" function with dummy data. However, the full flow of using the bind does not work.

Conclusion

Through careful planning, design, and implementation, we achieved our primary goal of creating a Minimum Viable Product (MVP). Highlights include a clean and intuitive user interface inspired by dark academia aesthetics, robust backend integration with APIs, and thoughtful consideration of accessibility and security. Despite challenges, we adapted through agile working and open communication, ensuring the project remained on track.