CSCI 3308 Group Project:
Cool Kids Book Application

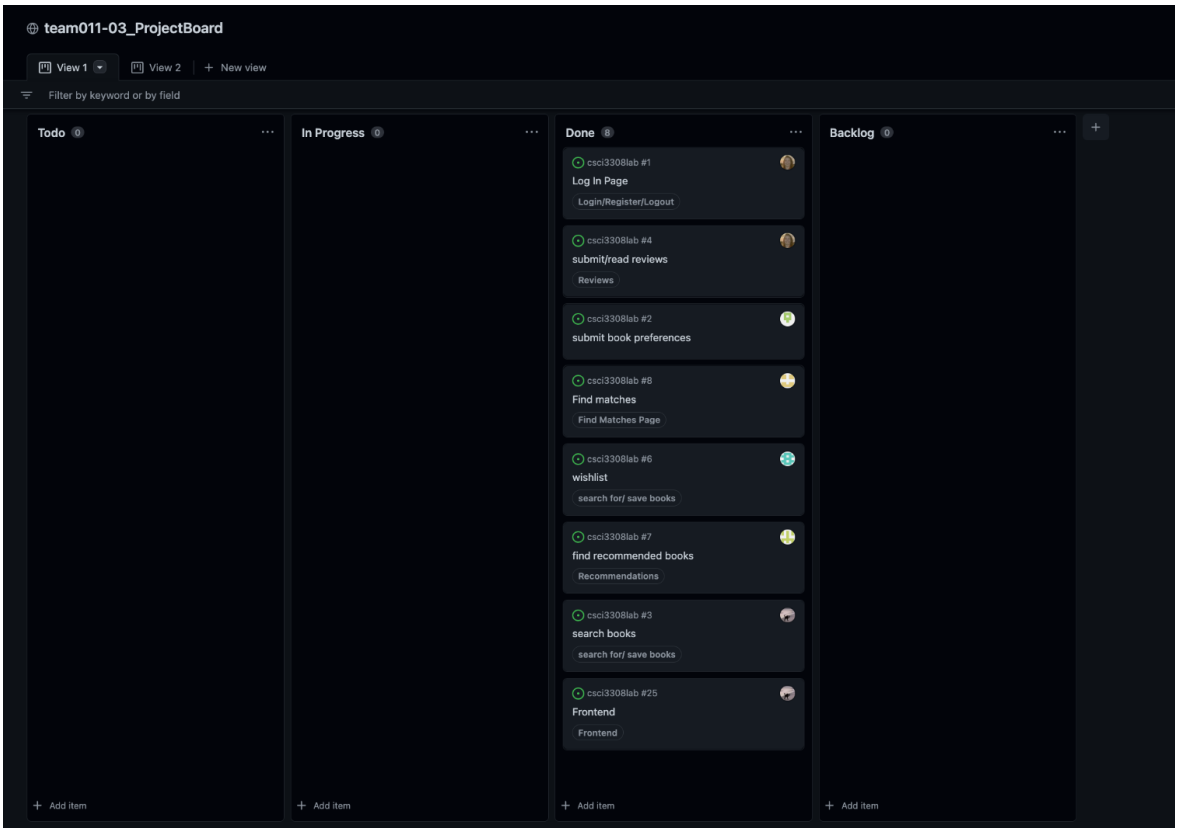
Madison Crouch, Nathaniel Lee, Avery McCauley, John Otterbein, Mallory Prescott, Lucca Quintela

I. **Project Description**:

Finding books to read and like-minded friends to discuss books with can be difficult. Our group has created a solution: an application that lets you find friends and book recommendations based on mutual interests. The Cool Kids Book Application is the ultimate app to connect with other readers, read and write reviews, and find new book recommendations. This application utilizes the Google Books API to give users the information they want in a seamless, visually appealing way, in addition to benefits of a social networking platform. Once users register and log in, they can read and submit book reviews, match with other users who have similar interests, search for books, and add books to their liked books list. Users can also find more information on books they discover through the corresponding Google Books link via the search feature. Another useful feature is the ability to create a wishlist of books that the user would like to read in the future. Users can log out and have their reviews, wishlist, liked books, and other information retained. The Cool Kids Book Application lets you keep track of all of your reading needs while connecting you with others!

II. **Project Tracker:**

Our project progress was tracked via this Github Project Board.



III. **Video:**

A video demonstration of our application can be found in our Github Repository here.

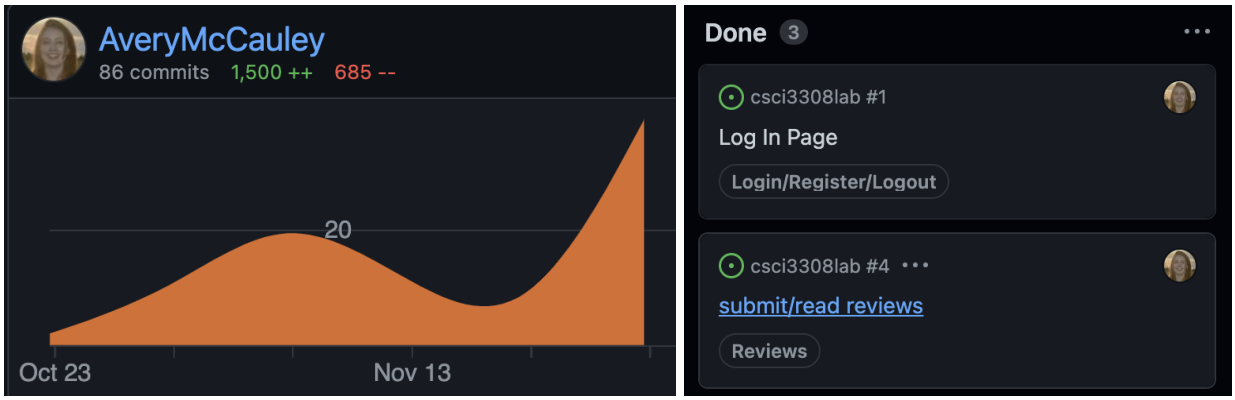
IV. Version Control Systems (VCS):

This Github repository was utilized to manage our project and includes the project source code, test cases, video demonstration, a README.md, a project board for project tracking, and all other relevant project documentation.

V. Contributions

A. Avery McCauley:

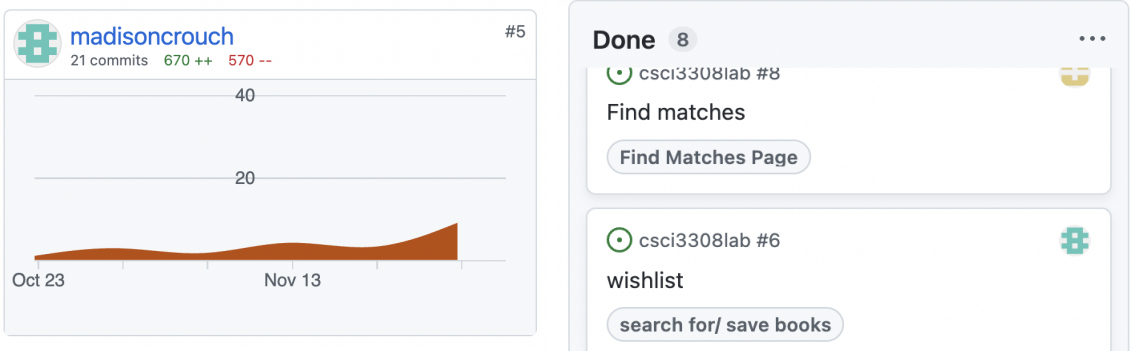
The features I worked on include the login, register, show reviews, and submit reviews pages. The review pages are populated with sample data for the purposes of the demo, and displays any new reviews submitted including the username (obtained from the session), book title, text review, and a numerical rating (1-5). The top five reviews are displayed at the top. The technologies I worked with include HTML, EJS, Docker, and PostgreSQL. As the designated scribe of the group, I also updated the release and meetings notes each week. Finally, I designed the slide presentation deck and the report outline.



Github Contributions and Project Board Completed Tasks

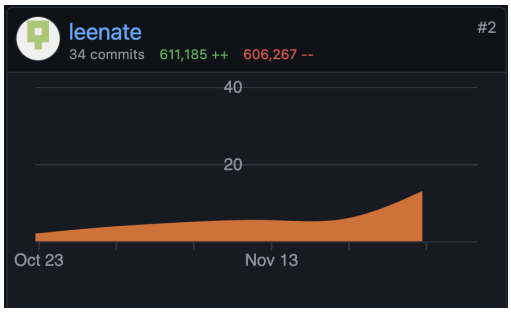
B. Madison Crouch:

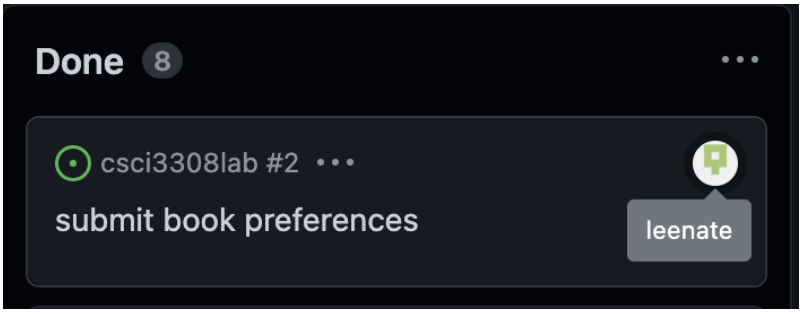
For our project I worked on the entirety of the wishlist page, including both the user interface and backend functionality. I used a get API to render the wishlist page. Within this API I queried the table user_to_book from our database which stores a user's wishlist books. The wishlist page displays all of the books a user has added to their wishlist iteratively on bootstrap cards. The information included in these cards is an image of the book and the book title as this was the relevant information we chose to store in our books database table. On each card there is also a "Remove Book" button which allows the user to remove a book from their wishlist. I implemented this functionality using a post API call which removes the book from user_to_books and redirects to the updated wishlist page to reflect this removal.



C. Nathaniel Lee:

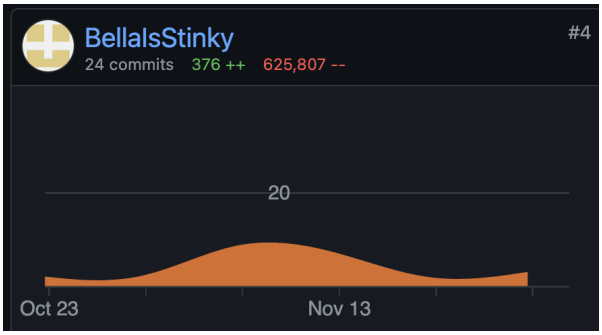
The work that I did was on the liked books page and the backend. I created the ejs and server side javascript for this page. The server-side javascript queried both the database and the external google books api using axios in order to populate the liked books on the page as they are added, and submit the liked books and the associated user using session variables to the database. Additionally, I created the docker-compose, git/.gitignore, and node configuration documents in order to get the website up and running. Also, I created the schema for the database (all tables and referential constraints) being sure to implement the specific business rules for our use case.

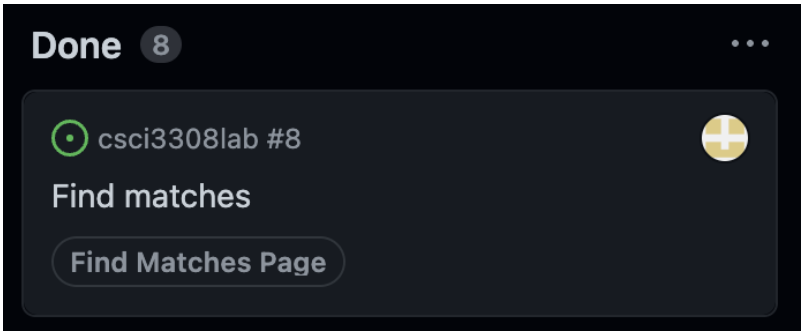




D. Lucca Quintela:

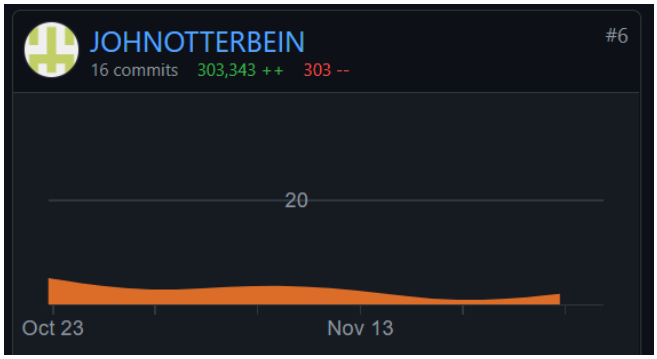
My job was to work on the find matches and friends page in its entirety, from backend to frontend. I created a get API to query for both matches and your current friends, as well as a post API to add new friends for the current user. The get API will find you relevant matches based on each of your book preferences and then remove any friends you add from that query to avoid duplicates. Throughout this project, I worked with Javascript, EJS, Docker, HTML, CSS and PostgreSQL.





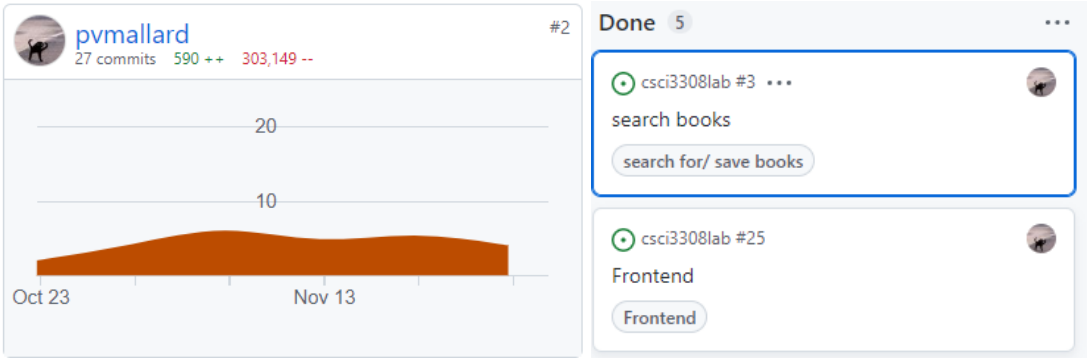
E. John Otterbein:

I was in charge of the functionality of the recommendation, including its operations to search the API, display the resulting information, and to have options to see the book when you click on it. The recommendation page by default will display five books and their corresponding covers, titles, authors, descriptions, alongside a button. The button redirects the user to the external API for more information on that book. For technologies, I worked with HTML, CSS, JS and EJS, Docker, and PostgreSQL.



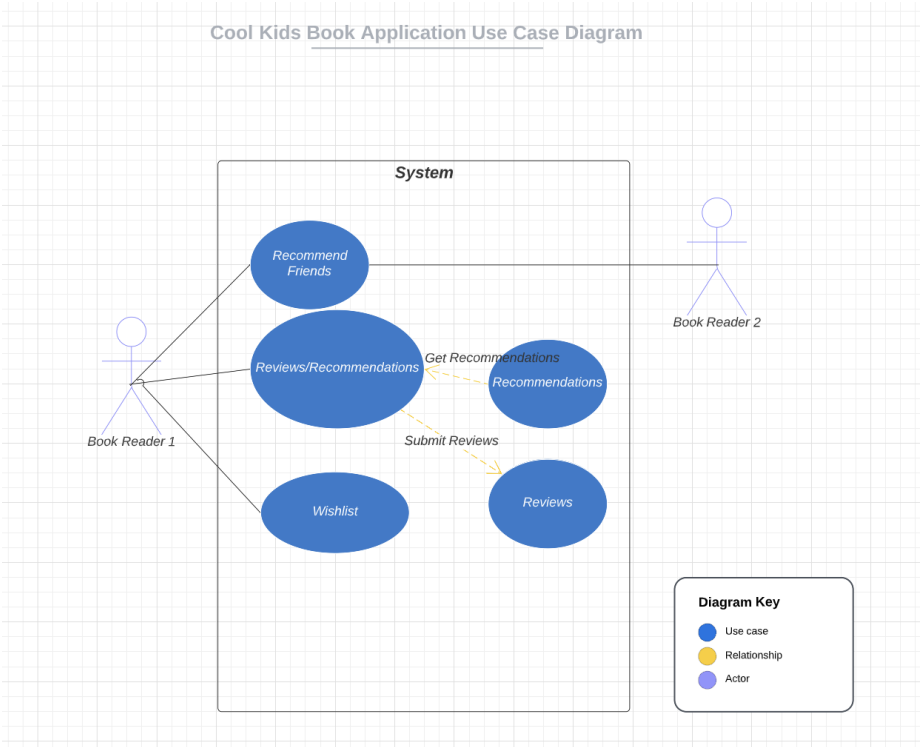
F. Mallory Prescott:

I was in charge of the functionality of the search page, including its operations to search the API, display the resulting information, and to have options to add or remove the shown books from the user's wishlist database. The search page by default will display ten books and their corresponding covers, titles, authors, descriptions, alongside two buttons. The first button redirects the user to the external API for more information on said book, while the second either adds or removes the books from their wishlist. The front-end css was another feature I worked on, just to make sure the styling came out how we wanted and that the display was fairly user-friendly. For technologies, I worked with HTML, CSS, JS and EJS, JSON, Docker, and PostgreSQL.



Github Contributions and Project Board Completed Tasks

VI. Use Case Diagram



VII. Test Results

Test Case #1: Login

User Acceptance Test Cases Description:

1. As a user, I should not be able to log in with a blank field. The data will consist of two fields: a blank username and/or a blank password. The user activity will include leaving either or both fields empty resulting in them being prompted to try again.
2. As a user, I should not be able to log in with an unregistered account. The data will consist of two fields: an invalid username and a password. The user activity will include trying to log in with an invalid username and/or password and then being prompted to try again or to register.
3. As a user, I should be able to log in with a registered account. The data will consist of two fields: a valid username and a valid password. The user activity will include trying to log in with a registered account and being redirected to their book preferences page.
4. As a user, I should be able to remain logged in unless I log out.

Test Data:

1. Username - i.e. Bob, ____, unregistered username
2. Password - i.e. BobsPassword, _____, 123abc

Expected Results:

1. If either or both fields are empty, then the page should state that both fields are required.
2. If the username is a registered user but the password is incorrect, the page should display a message saying the credentials were not valid, and allow the user to try to log in again.
3. If the username is a registered user and the password is correct, the user should be redirected to their home page.
4. If the username is not a registered user, they should be redirected to the registration page.

Test Results:

1. When the user attempts to login without filling out username or password fields, there is an error message and they are not be logged in.
2. When the user enters the correct username but an incorrect password, there is an error message and they are not be logged in.
3. When both the username and password entered by the user are correct, the user is logged in and both of these values are stored in session variables.
4. When the user enters a username which is not in the database, they are redirected to the register page.

Observations:

1. What are the users doing?
   Users enter a memorable username and password in order to login to the application
2. What is the user's reasoning for their actions?
   Users want to have an account with the application so data such as liked books, wishlist, friends/matches, and recommendations will appear in the application each time they login
3. Is their behavior consistent with the use case?
   yes
4. If there is a deviation from the expected actions, what is the reason for that? n/a
5. Did you use that to make changes to your application? If so, what changes did you make?
   No changes made based on user observations.

Test Case #2: Find Matches

User Acceptance Test Cases Description:

1. Should be able to find relevant matches based on book preferences
2. Should be able to add any matches as a friend
3. Should be able to look through list of friends

Test Data:

1. Usernames
2. Previous Books of Users
3. Previously Read Authors of Users
4. Books on wishlist

Expected Results:

1. Whether the matches are actually relevant
2. Whether the friends are properly added to the friends list
3. Whether the user data is correct

Test Results:

1. Due to the small number of users currently existent in our database as well as the lack of a genre category in the google books api, ensuring relevant matches is slightly difficult.
2. When the user clicks on a friend to add, the friends username is removed from matches and instead appears in the friends category.
3. When the user navigates to another page and then returns to the friends page, the information is consistent with their past activity on this page.

Observations:

1. What are the users doing?
   Users navigate to the matches/friends page in order to find other users with similar book interests
2. What is the user's reasoning for their actions?
   To connect with people who like the same books that they do - like minded people to connect with.
3. Is their behavior consistent with the use case?
   Yes, some users also requested a remove friend functionality as well as a way to message friends.
4. If there is a deviation from the expected actions, what is the reason for that?
   Ideally, users would be able to interact with the friends they add on the application, as well as remove friends they no longer wish to interact with.
5. Did you use that to make changes to your application? If so, what changes did you make?
   In the future we may choose to add these functionalities to our application however, given the time frame of this project we chose to sacrifice these changes for more fundamental functionality of our application.

Test Case #3: Submit Reviews

User Acceptance Test Cases Description:

1. Have a field where the user can submit their reviews. Have these reviews successfully submitted to the db and viewable.

Test Data:

1. Different users
2. Strings of varying length
3. Different books associated with reviews

Expected Results:

1. If either field is empty the submit should fail and output a warning to the user.
2. If both fields are completed, the review should be submitted and accessible in the db.
3. Reviews are viewable either by searching for a book or looking at a user's history.
4. If there is a semantic error, submit should fail and a warning message should be output to the user.

Test Results:

1. When the user attempts to submit a review without filling out all fields, the review is not submitted and there is an error message
2. When the user appropriately fills out all review fields, the review is submitted and the user can scroll to the bottom of the reviews page to see their most recent review
3. When the user searches for a book, reviews for the book are not displayed. User history is also not available to view
4. No semantic error occurs when a user submits a review

Observations:

1. What are the users doing?
   Users submit reviews of books - typically books that they liked to read. Users also view reviews submitted by other users
2. What is the user's reasoning for their actions?
   Users want to submit such reviews primarily so their friends can see which books they liked or disliked as well as view reviews made by others so they can potentially find new books to read
3. Is their behavior consistent with the use case?
   Yes
4. If there is a deviation from the expected actions, what is the reason for that?
   Suggestions for the reviews page were to include a timestamp on each review so it is more clear to the user that the reviews are listed in order from oldest to most recent
5. Did you use that to make changes to your application? If so, what changes did you make?
   If given more time on our application we would likely add timestamps to each review listed on the page

Test Case #4: Add Wishlist

User Acceptance Test Cases:

1. Should be able to search for any book by title, author, etc. and select to add the book to the user's wishlist
2. If the book has already been added to the wishlist and the user attempts to add it again, a message should inform them it already exists in their wishlist
3. Once the user adds a book to their wishlist, they should be redirected to the wishlist to confirm the addition

Test Data:

1. Search for different titles, authors, as well as search which contains both a book title and author name
2. Attempt to add new books to the wishlist as well as books already contained in the user's wishlist

Expected Results:

1. If the user types in the title, author, or both the page should display the correct results
2. If the user adds a new book to their wishlist, the addition is successful and redirects them to the wishlist page
3. If the user attempts to add an existing book to their wishlist, an error message will appear letting them know that this book is already in their wishlist

Test Results:

1. Search bar works as expected
2. Adding a new book to the wishlist results in a redirect to the wishlist page which displays the user's wishlist
3. Attempting to add an existing book to the wishlist displays an error message and does not change the wishlist page

Observations:

1. What are the users doing?
   Users search for books based on a specific title or an author they are interested in
2. What is the user's reasoning for their actions?
   Users want to find books that they are interested in reading but have not yet had the opportunity to do so. Saving these books allows them to come back later and potentially purchase/download the book(s)
3. Is their behavior consistent with the use case?
   yes
4. If there is a deviation from the expected actions, what is the reason for that?
   n/a
5. Did you use that to make changes to your application? If so, what changes did you make?
   No changes made based on user observations

VIII. Deployment

Due to deployment issues with the CU network, our group continues to host the application on local host (port 3000). The Docker application must be running to access the app. The user must download the Project Code folder from the repository and run `docker compose up` in a shell of their choice. The application is then accessed using Google Chrome by typing “127.0.0.1:3000” in the address bar.