Use case analyses and descriptions:
https://docs.google.com/document/d/1HdPGJvSzrj295bqhX4X8VhrdzUbwLkhbzCzkYwx_PRQ/edit?tab=t.0

Ty: 1, 2, 3, 4, 5, 6
Benjamin: 7, 8, 9, 10, 11, 12
Sky: 14, 16, 17, 18, 19, 20
Jaden: 13, 15, 21, 22, 23, 24

Use Case UC1: Allow users to sign up with email, Google, or social media accounts

Algorithm 1: The **SignUp()** up function

```
   Input: none
   Output: none
1.1  Prompt user to choose sign-up method (Email, Google, Social Media)
1.2  method ← GetUserInput()
1.3  if method == "Email" then
1.4  |  Prompt for email and password
1.5  |  email ← GetUserInput()
1.6  |  password ← GetUserInput()
1.7  |  if email contains "@" and password meets criteria then
1.8  |  |  Create account with email and password
1.9  |  |  Print "Account created successfully"
1.10 |  else
1.11 |  |  Print "Invalid email or password"
1.12 else if method == "Google" or method == "Social Media" then
1.13 |  Redirect to chosen platform's sign-in
1.14 |  if authentication successful then
1.15 |  |  Retrieve user information
1.16 |  |  Create account with retrieved information
1.17 |  |  Print "Account created successfully"
1.18 |  else
1.19 |  |  Print "Authentication failed"
1.20 else
1.21 |  Print "Invalid choice"
```

Use Case UC2: Password and username to validate registered accounts

Algorithm 2: The **LogIn()** function

```
   Input: none
   Output: none
2.1  Prompt for username and password
2.2  if credentials are valid then
2.3  |  Print "Login successful"
2.4  else
2.5  |  Print "Invalid credentials"
2.6  |  Offer options: Retry, Forgot Password, Forgot Username
2.7  |  choice ← GetUserInput()
2.8  |  if choice is "Forgot Password" then
2.9  |  |  Handle password reset
2.10 |  else if choice is "Forgot Username" then
2.11 |  |  Handle username recovery
2.12 |  else if choice is "Retry" then
2.13 |  |  LogIn()
```

Use Case UC3: Create profile (profile pic/bio)

Algorithm 3: The **UpdateProfile()** function

```
   Input: none
   Output: none
3.1 if user wants to update bio then
3.2     Prompt for bio (max 400 characters)
3.3     if bio length ≤ 400 then
3.4         Save bio
3.5         Print "Bio updated"
3.6     else
3.7         Print "Bio too long"
3.8 if user wants to update profile picture then
3.9     Prompt to upload photo
3.10    if photo meets requirements then
3.11        Save photo
3.12        Print "Profile picture updated"
3.13    else
3.14        Print "Invalid photo"
```

Use Case UC4: Ability to create book club group and join

Algorithm 4: The **CreateBookClub()** function

```
   Input: none
   Output: none
4.1 Prompt for club name
4.2 if name is empty then
4.3     Print "Club name is required"
4.4     return
        // Exit function
4.5 Prompt for bio (optional, max 1000 characters)
4.6 Prompt to upload photo (optional)
4.7 if photo is invalid or not provided then
4.8     Use default photo
4.9 Save club details (name, bio, photo, contact info, genre)
4.10 Print "Book club created successfully"
```

Use Case UC5: Share pictures with the group

Algorithm 5: The **SharePicture()** function

```
   Input: none
   Output: none
5.1 if group allows picture sharing then
5.2     Prompt to select photo
5.3     Prompt for caption (optional)
5.4     if photo upload is successful then
5.5         Print "Picture shared successfully"
5.6     else
5.7         Offer options: Retry or Cancel
5.8         if user selects "Retry" then
5.9             SharePicture()
5.10 else
5.11    Print "Picture sharing is disabled in this group"
```

Use Case UC 6: Ability to block/unblock users from joining the group

Algorithm 6: The **ManageBlockedUsers()** function

```
    Input: none
    Output: none
6.1 Prompt admin to choose action: Block or Unblock
6.2 action ← GetUserInput()
6.3 if action is "Block" then
6.4 |   Prompt for username to block
6.5 |   Confirm action
6.6 |   if confirmed then
6.7 |   |   Add user to block list
6.8 |   |   Print "User blocked"
6.9 else if action is "Unblock" then
6.10|   Print list of blocked users
6.11|   Prompt for username to unblock
6.12|   Confirm action
6.13|   if confirmed then
6.14|   |   Remove user from block list
6.15|   |   Print "User unblocked"
6.16 else
6.17|   Print "Invalid choice"
```

Use Case UC 7: Set group moderators

Algorithm 7: The **setGroupModerators()** function

```
    Input: c - User's credentials
    Output: None
7.1 moderators←Hash Table storing group's moderators and their permissions
7.2 if c=OwnerCredentials then
7.3 |   Change UI state to include a role form on the club's member list
7.4 |   form ← User selects "Make member a moderator" beside each member they
    |     want to be a moderator and then sets their permissions
7.5 |   User selects "Save changes"
7.6 |   Prompt "Confirm changes"
7.7 |   if User selects "Yes" to confirm the changes then
7.8 |   |   foreach user, permissions ∈ form do
7.9 |   |   |   add KVPair(user : permissions) to moderators;
7.10|   |   Print "Changes Saved"
7.11|   else
7.12|   |   Print "Changes not saved"
```

Use Case UC 8: Borrow or buy books

Algorithm 8: The **buyOrBorrowBooks()** function

```
    Input: None
    Output: None
8.1 User selects the book they want to read from the Library Page
8.2 User selects "Find a copy" on Book Page
8.3 Offer options: "Add to cart", "Libraries", "More options"
8.4 choice ← getUserInput()
8.5 if choice="Add to cart" then
8.6 |   shoppingCart← User's Shopping Cart
8.7 |   book← Book user wants
8.8 |   add book to shoppingCart
8.9 |   if user goes to their shoppingCart then
8.10|   |   Display cart
8.11|   |   if user selects "Checkout" then
8.12|   |   |   prompt user for their credit card information and address
8.13|   |   |   creditCard← getUserInput()
8.14|   |   |   address← getUserInput()
8.15|   |   |   if validPaymentMethod(creditCard) then
8.16|   |   |   |   orderNumber←processPayment(creditCard, Address)
8.17|   |   |   |   Print "Received Order: orderNumber"
8.18|   |   |   else
8.19|   |   |   |   Print "Invalid Payment Method"
8.20|   |   |   |   Redirect user to enter their credit card information and address
    |   |   |   |     again
8.21| else if choice="Libraries" then
8.22|   |   Redirect user to worldcat/book
8.23| else if choice="More Options" then
8.24|   |   if user selects Amazon then
8.25|   |   |   Redirect user to amazon/book
8.26|   |   else if user selects Barnes and Noble then
8.27|   |   |   Redirect user to barnesandnoble/book
```

Use Case UC9: Search for books

Algorithm 9: The **searchBooks()** function

```
    Input: None
    Output: List of books
9.1 Display input fields for title, author, ISBN
9.2 title←getUserInput()
9.3 isbn←getUserInput()
9.4 author←getUserInput()
9.5 if isbn != null and isbn is not a number then
9.6 |   Print "Invalid ISBN"
9.7 else
9.8 |   query←Query()
9.9 |   if isbn != null then
9.10|   |   buildQuery(isbn, query)
9.11|   if title != null then
9.12|   |   buildQuery(title, fuzzySearch=True, query)
9.13|   if author != null then
9.14|   |   buildQuery(author, fuzzySearch=True, query)
9.15|   books←queryDatabase(query).limit(1000)
9.16|   if length(books)=0 then
9.17|   |   Print "No results"
9.18|   else
9.19|   |   Display books
```

Use Case UC10: Set a group name for the book club

Algorithm 10: The **setGroupName()** function

```
    Input: username - username of the user making the change
    Output: None
10.1 moderators←Hash Table storing group's moderators and their permissions
10.2 if "SET NAME" in moderators[username] then
10.3 |   Change BookClub page to have a "Edit club name" button
10.4 |   User clicks "Edit club name"
10.5 |   Prompt user to enter a name
10.6 |   name←getUserInput()
10.7 |   User clicks "Update" button
10.8 |   if name not taken and is not inappropriate and has valid characters then
10.9 |   |   Write name to the database
10.10|   else Redirect user to page to enter a name
10.11|   Print "Invalid name: try again"
```

Use Case UC11: Set the current book

Algorithm 11: The **setCurrentBook()** function

```
    Input: username - username of the user making the change
    Output: None
11.1 moderators←Hash Table storing group's moderators and their permissions
11.2 if "SET BOOK" in moderators[username] then
11.3     Change BookClub page to have a "Create poll" button
11.4     User clicks "Create poll"
11.5     readingList←Set of books defined by the club members
11.6     A poll is created
11.7     Club members use the poll to vote for a book from readingList
11.8     The poll closes after all club members vote or after 24 hours
11.9     newBook←null
11.10    max←0
11.11    foreach votes, book ∈ poll do
11.12        if votes > max then
11.13            max←votes
11.14            newBook←book
11.15    Write newBook to the database
11.16    Print "Club's book set to book"
```

Use Case UC12: Set group type (by genre)

Algorithm 12: The **setGroupType()** function

```
    Input: username - username of user making the change
    Output: None
12.1 moderators←Hash Table storing group's moderators and their permissions
12.2 if "SET TYPE" in moderators[username] then
12.3     Change BookClub page to have a "Edit club type" button
12.4     User clicks "Edit club type"
12.5     Prompt user to enter a genre
12.6     genre←getUserInput()
12.7     if User selects a genre from the drop down menu then
12.8         User clicks "Update" button
12.9         Write genre to the database
12.10        Print "Club type updated"
```

Jaden 13:

Use Case UC13: Book recommendations

Algorithm 13: the **BookRecommendations()** function

Input: none
Output: *bookRecs*-book recommendations

```
13.1 if User Not Logged In then
13.2     User will be prompted to login
13.3 else if User is in home page then
13.4     User will be shown a feed of reccomended books based on books read and clubs
           joined
```

Sky:

Use Case UC14: Add books to reading club list

Algorithm 14: The **addBooksToClubList()** function

Input: *book*—a book to be added, *club*—a reading club, and *user*—a user of the platform
Output: *True* if the book was successfully added, *False* otherwise

```
14.1  if NOT (user is Logged In AND user is Admin) then
14.2      return false
14.3  if book IS NOT NULL then
14.4      SEARCH_DATABASE(book)
14.5  else
14.6      print("Enter book details manually.");
14.7      userResponse ← GET_USER_INPUT();
14.8      book ← FIND_BOOK(userResponse);
14.9  if club.ReadingList CONTAINS book then
14.10     print("Book is already in list.  Check recommendations for other
            books!");
14.11     return false
14.12 Add book to club.ReadingList;
14.13 SAVE_TO_DATABASE(club.ReadingList);
14.14 return true;
```

Jaden 15:

Use case UC15: Create polls

Algorithm 15: the **Polls()** function

Input: *votes*-votes to be tallied for polls
Output: *poll*-chart of the poll and its results

15.1 **if** *User is admin in club* **then**
15.2 User will see option for starting a poll
15.3 **else**
15.4 User is not able to start poll
15.5 **if** *admin selects "Start Poll"* **then**
15.6 Window to create poll will pop up.
15.7 Window will include options for a title, options, and duration of poll.
15.8 Once these fields are entered members of club will be able to vote.
15.9 **NotificationForDueDates()** will be called to inform members of poll. The votes will be counted as long as it is done within the duration of the poll.
15.10 Once all votes are in poll results are posted to club.

Sky:

Use Case UC16: Remove/add books from club reading list

Algorithm 16: The **deleteBooksFromClubList()** function

Input: *book*—a book to be removed, *club*—a reading club, and *user*—a user of the platform
Output: *True* if the book was successfully removed, *False* otherwise

16.1 **if** *NOT (user is LoggedIn* ***AND*** *user is Admin)* **then**
16.2 **return false**
16.3 **if** *club.CurrentBook IS book* **then**
16.4 `print("This book is currently being discussed. Are you sure you want to proceed?");`
16.5 *userResponse* ← `GET_USER_INPUT();`
16.6 **if** *userResponse IS 'NO'* **then**
16.7 **return false**
16.8 **else**
16.9 `Remove book from club.ReadingList;`
16.10 `SAVE_TO_DATABASE(club.ReadingList);`
16.11 `print("Book removed from the reading list!");`
16.12 **return true**

Use Case UC17: Set challenges/read by dates (progress)

Algorithm 17.1: The `setChallenges()` function

Input: A user, `user`
Output: Returns `true` if the reading challenge was successfully set, `false` otherwise

```
17.1.1  Challenge_event ← READING_GOAL_PAGE CREATE CHALLENGE;
17.1.2  if user is NOT Logged In then
17.1.3      print("You need to be logged in to access this feature");
17.1.4      return false;
17.1.5  print("Set event as personal goal or club goal:");
17.1.6  userResponse ← GET_USER_INPUT();
17.1.7  if userResponse IS 'club goal' then
17.1.8      if user is NOT Admin AND user DOES NOT have Special
             Privileges then
17.1.9          print("You need to be an admin to set a club
                 challenge");
17.1.10         return false;
17.1.11 WAIT FOR READING_GOAL_PAGE SUBMIT Challenge_event;
17.1.12 Pages_to_read ← Challenge_event.GET("Pages To Read");
17.1.13 Frequency ← Challenge_event.GET("Frequency of Challenge");
17.1.14 Start_date ← Challenge_event.GET("Start Date");
17.1.15 End_date ← Challenge_event.GET("End Goal Date");
17.1.16 user.challenge ← Challenge_event;
17.1.17 READING_GOAL_PAGE add challenge_event to active list;
17.1.18 READING_GOAL_PAGE DISPLAY Challenge_event;
17.1.19 print("Your reading challenge has been set!");
17.1.20 return true;
```

Algorithm 17.2: The `notifyChallenges()` function

Input: A user, `user`
Output: None

```
17.2.1  if userchallenge IS NOT NULL then
17.2.2      if user notifications IS ON then
17.2.3          READING_GOAL_PAGE notify("You have a reading goal in
                 progress!");
17.2.4          if user daily_pages ≥ user challenge "Pages To Read"
                 then
17.2.5              READING_GOAL_PAGE notify("You have completed your
                     reading goal for today!  Good work!");
17.2.6          if user challenge "End Goal Date") IS GET_SYSTEM_DATE()
                 then
17.2.7              READING_GOAL_PAGE notify("Your reading challenge
                     has ended!  Great job!");
```

Use Case UC18: Read books online (with ability to bookmark, make notes, highlight, set chapter length goals, personalize font, size, colors, etc)

Algorithm 18: The **readBookOnline()** function

Input: A logged-in user, user, and a purchased book in the user's bookshelf, book
Output: None

```
18.1  if SYSTEM_CONNECTION ≠ NULL or book is downloaded then
18.2      Session ← READER load Session(book);
18.3      if Session = NULL then
18.4          book.current_page ← 1;
18.5          Session ← READER create Session(book, book.current_page);
18.6      Session open Book(book, book.current_page);
18.7      while Session is Open do
18.8          currentPage ← Session get Current Page;
18.9          if book.current_page ≠ currentPage then
18.10             book.current_page ← currentPage;
18.11             Session save Progress(book.current_page);
18.12         if USER_ACTION = "Bookmark" then
18.13             bookmark ← Session add Bookmark to current Page;
18.14         if USER_ACTION = "Add Note" then
18.15             Note ← GET_USER_INPUT("Enter your note:");
18.16             Session add Note to currentPage;
18.17         if USER_ACTION = "Highlight" then
18.18             Highlight ← GET_USER_SELECTION("Select text to highlight:");
18.19             Session add Highlight to current page);
18.20         if USER_ACTION = "Customize View" then
18.21             customizationOptions ← GET_USER_INPUT("Choose font, size, or
                   color settings:");
18.22             Session apply Customizations;
18.23     READER save Session;
```

Use Case UC19: Find local book clubs nearby (register your book club online/on the app)

Algorithm 19: The **findOrRegisterBookClub()** function

Input: A logged-in user, **user**
Output: Displays nearby book clubs or registers a new book club

```
19.1  if user is Logged In then
19.2      if USER_ACTION = "Find Book Clubs" then
19.3          if user.location = NULL then
19.4              user.location ← GET_USER_LOCATION();
19.5          bookClubs ← Search nearby clubs within 25 miles of user.location);
19.6          if bookClubs is Empty then
19.7              DISPLAY MESSAGE("No book clubs found nearby.");
19.8          else
19.9              DISPLAY bookClubs;
19.10     else if USER_ACTION = "Register Book Club" then
19.11         if user does NOT have Book Club then
19.12             DISPLAY MESSAGE("Please create and name a book club first.");
19.13             return;
19.14         clubInfo ← GET_USER_INPUT("Enter book club information:");
19.15         Register book club(user, clubInfo);
19.16         DISPLAY MESSAGE("Your book club has been successfully
               registered!");
19.17 else
19.18     DISPLAY MESSAGE("Please log in to access this feature.");
```

Use Case UC20: Submit club thoughts, opinions, reviews about the books as public notes/posts

Algorithm 20: The **submitPost()** function

Input: A logged-in user, `user`, and their draft post, `draftPost`
Output: The post is published to the public post tab

```
20.1  if user is Logged In then
20.2      if draft Post contains Inappropriate Language then
20.3          BLOCK POST;
20.4          DISPLAY MESSAGE("Post contains inappropriate language.");
20.5          return;
20.6      if draft Post contains a Spoiler then
20.7          FLAG draftPost as SPOILER;
20.8      if draft Post is Ready To Submit then
20.9          Publish draftPost to public tab;
20.10         DISPLAY MESSAGE("Post published successfully!");
20.11     else if USER_ACTION = "Save as Draft" then
20.12         SAVE draftPost as DRAFT;
20.13         DISPLAY MESSAGE("Draft saved successfully.");
20.14     else if USER_ACTION = "Delete Post" then
20.15         DELETE draftPost;
20.16         DISPLAY MESSAGE("Post deleted.");
20.17     else if USER_ACTION = "Suggest Spoiler" then
20.18         SUGGEST SPOILER(draftPost);
20.19         DISPLAY MESSAGE("Spoiler tag suggested.");
20.20 else
20.21     DISPLAY MESSAGE("Please log in to submit a post.");
```

Use Case UC21: Have private chat channels for book clubs to discuss books

Algorithm 21: the **PrivateChannel()** function

Input: *user*-user to make the channel, *bookClub* - book club that hosts private channel, *channelName* - name of the private channel
Output: *succesStatement*- lets user know they successfully created the channel

```
21.1  if the user is a club moderator they will see a "create channel" option in the menu
21.2  if User selects "Create Channel" then
21.3      user prompted to enter name of channel
21.4      if Channel name does not contain any hate speech detected then
21.5          channel is created, user is sent a success screen that channel was properly
              created, and user is prompted to invite club members if User selects "Add
              Members" then
21.6              pop up menu of all book club members is opened and admin can select
                  members to invite to channel
21.7          else
21.8              nothing
21.9      else
21.10         User is prompted to select a new name
```

Use Case UC22: Notifications for due dates

Algorithm 22: the **NotificationForDueDates()** function

Input: none
Output: *DueDate*- Informs user of impending due date

```
22.1  if If due date is just created then
22.2      Users of book club are notified of upcoming due date and any message that is
          sent out by club admin
22.3  else if Due date already created and one day before due then
22.4      Users are sent another notification of what is due and when
22.5  else if Day of due date then
22.6      users are again notified of due date and what needs to be done
```

Use Case UC23: Export reading data to external file

Algorithm 23: the **ExportDataToFile()** function

Input: *ReadingData* - user's reading data
Output: *File*- File that displays users reading data

23.1 **if** *user requests reading data* **then**
23.2 Compile all available reading data. Display data for:NumBooksRead, TimeToFinish, GenresRead, AverageTimeToFinishBook,ClubsJoined, and ClubsLeft.
23.3 **NumBooksRead** will be available in bar graph form or line chart form.
23.4 **TimeToFinish** will be available in histogram and box plot
23.5 **GenresRead** will be available in bar graph and pie graph.
23.6 **AverageTimeToFinishBook** will be available in bar graph and line chart.
23.7 **ClubsJoined** and **ClubsLeft** will be available in bar graph and line chart forms.

Use Case UC24: Calendar features to see book club dates, deadlines, and community events

Algorithm 24: the **CalendarFeatures()** function

Input: none
Output: *Calandar*- Calendar that displays due dates and events

24.1 **if** *User not logged in* **then**
24.2 User will be prompted to sign in before viewing calendar details
24.3 **else**
24.4 User is able to view calendar and see events, due dates, and other notifications for the month
24.5 **if** *User selects a day* **then**
24.6 User will be prompted to insert details to make an event/ due date for that day
24.7 **if** *User Selects due date/event/deadline* **then**
24.8 Details of event are displayed like: what is due or what is happening and what club has posted this event