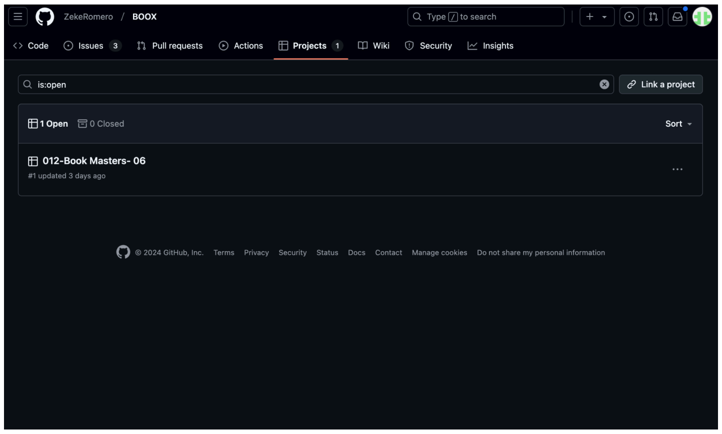
BOOX

Group 012-06

Team Members: (Contributions are listed at the end of the document)

- Zeke Romero, ZekeRomero,
- Charlie Hewitt, hewitt89
- Neerja Akruwala, neak8757
- Ben Floyd, befl6798
- Ayush Khadka, ayushkhadka514
- Brandon Reeves, brre8838

Project Tracker: Project Tracker



Repository Link: https://github.com/ZekeRomero/BOOX

Deployment: https://boox.onrender.com (Deployed to the internet using Render)
Slide Presentation: Presentation Link
VCS/ Video Link: https://youtu.be/DLZJ3ypAByI (also present in the readme file in our GitHub repository)

Project Description:

Boox is a multi-user book-sharing platform designed to connect book enthusiasts from all around the globe by creating a welcoming space for sharing, discovering, and discussing their love for literature. Users can explore a vast collection of books through an integrated API key, granting them access to an extensive range of titles to explore, ensuring something for every taste and preference.

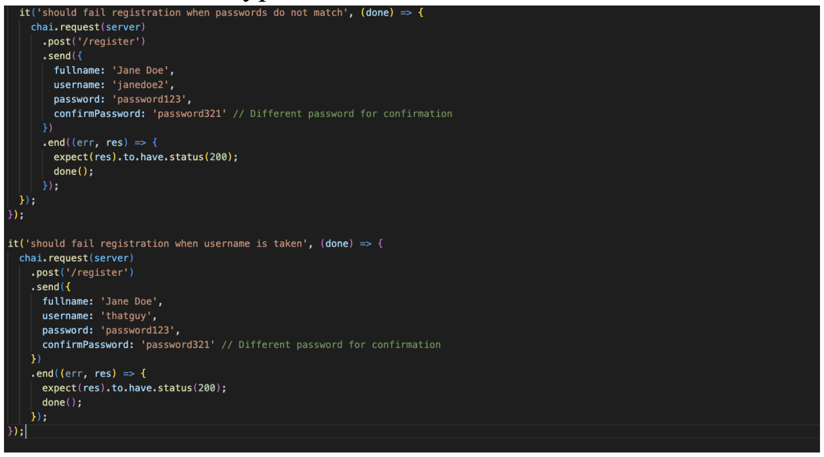
Upon joining, users gain access to a variety of engaging features. When creating an account, they can select a unique, displayable username and choose a profile picture from our fifteen curated options. Users can also connect with friends who share similar literary interests, making the experience more social and personalized. By searching for a novel, they can view detailed information about the book, such as the author, release year, and a brief summary, excluding the book's full content. Additionally, users can organize their reading lists by sorting titles into collections such as their favorite books or possible future reads for ease of access. Beyond collections, users can review and share their thoughts on books through a dedicated reviews page visible to the entire community. Boox is more than a platform for individual discovery—it's a hub for fostering meaningful connections among readers with shared passions.

Test Cases: UAT Plan Linked In Milestone Submissions Folder

https://github.com/ZekeRomero/BOOX/blob/main/MilestoneSubmissions/UAT.pdf

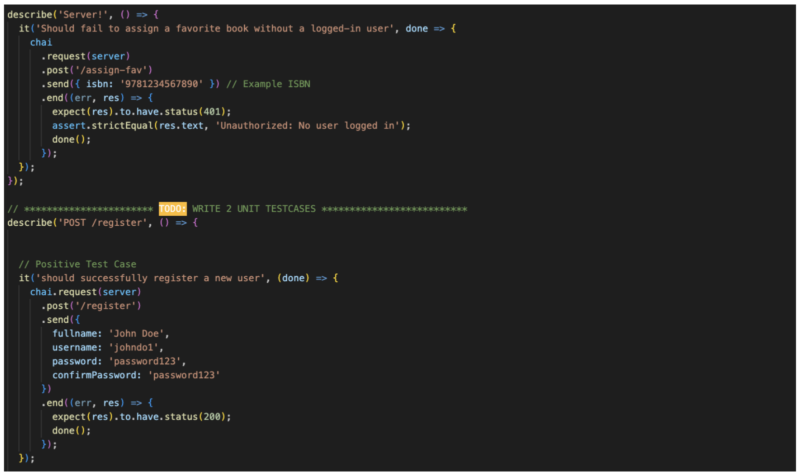
Test Case Descriptions:

1. When a user is registering for an account in order to access the rest of the website's features, if the username chosen is preexisting in the database, it will not allow the user to register with that username and will prompt them to pick a different one. The database can only have a username be used once because it is used as the primary key for the database. So if two users are able to login with the same username, it could cause some security issues within the website itself.
2. If a user is not logged in, they will not be able to select a book to add to a collection. Due to the use of sessions, a user is meant to login before they can change anything about the profile. If the user is logged out, there technically is no profile, thus they would not be able to change anything. Thus if a user is not logged in, the add to collections button should be hidden, and a user will not be able to access that feature until they are logged into the database.
3. When a user registers for an account, their information should automatically be added to the database allowing them to log in during the same session. When a user creates an account, they want that information to be saved so that they are actually able to log in and access all the features. If this does not work, the functionality of the entire website is endangered because we would not be able to access it.
4. When a user is registering for an account it will ask the user to enter a password, and then re-enter it to confirm. If the two passwords do not match, it will not allow the user to register. This is a simple security and authentication protocol most websites use because a password is usually hidden and necessary to log in. A user is told to confirm the password to make sure it is exactly what they want as it will not be able to be seen when typed. This ensures that a user is properly able to log in with one password instead of accidental typo's



```
  it('should fail registration when passwords do not match', (done) => {
    chai.request(server)
      .post('/register')
      .send({
        fullname: 'Jane Doe',
        username: 'janedoe2',
        password: 'password123',
        confirmPassword: 'password321' // Different password for confirmation
      })
      .end((err, res) => {
        expect(res).to.have.status(200);
        done();
      });
  });
});

it('should fail registration when username is taken', (done) => {
  chai.request(server)
    .post('/register')
    .send({
      fullname: 'Jane Doe',
      username: 'thatguy',
      password: 'password123',
      confirmPassword: 'password321' // Different password for confirmation
    })
    .end((err, res) => {
      expect(res).to.have.status(200);
      done();
    });
});
```



```
describe('Server!', () => {
  it('Should fail to assign a favorite book without a logged-in user', done => {
    chai
      .request(server)
      .post('/assign-fav')
      .send({ isbn: '9781234567890' }) // Example ISBN
      .end((err, res) => {
        expect(res).to.have.status(401);
        assert.strictEqual(res.text, 'Unauthorized: No user logged in');
        done();
      });
  });
});

// *********************** TODO: WRITE 2 UNIT TESTCASES **************************
describe('POST /register', () => {


  // Positive Test Case
  it('should successfully register a new user', (done) => {
    chai.request(server)
      .post('/register')
      .send({
        fullname: 'John Doe',
        username: 'johndo1',
        password: 'password123',
        confirmPassword: 'password123'
      })
      .end((err, res) => {
        expect(res).to.have.status(200);
        done();
      });
  });
```

<u>Expected Outputs:</u>

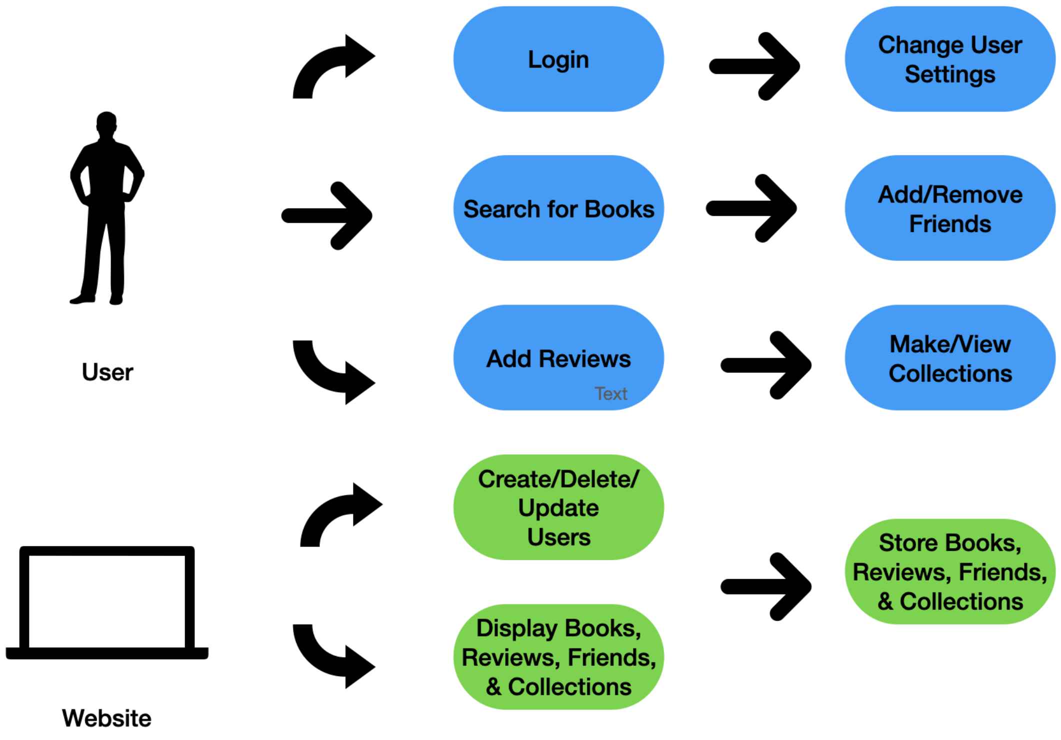
```
web-1 | ✔ should fail registration when username is taken
web-1  |     ✔ Should fail to assign a favorite book without a logged-in user
web-1  |
web-1  |   POST /register
web-1  |     ✔ should successfully register a new user (74ms)
web-1  |     ✔ should fail registration when passwords do not match
```

<u>Observations:</u>

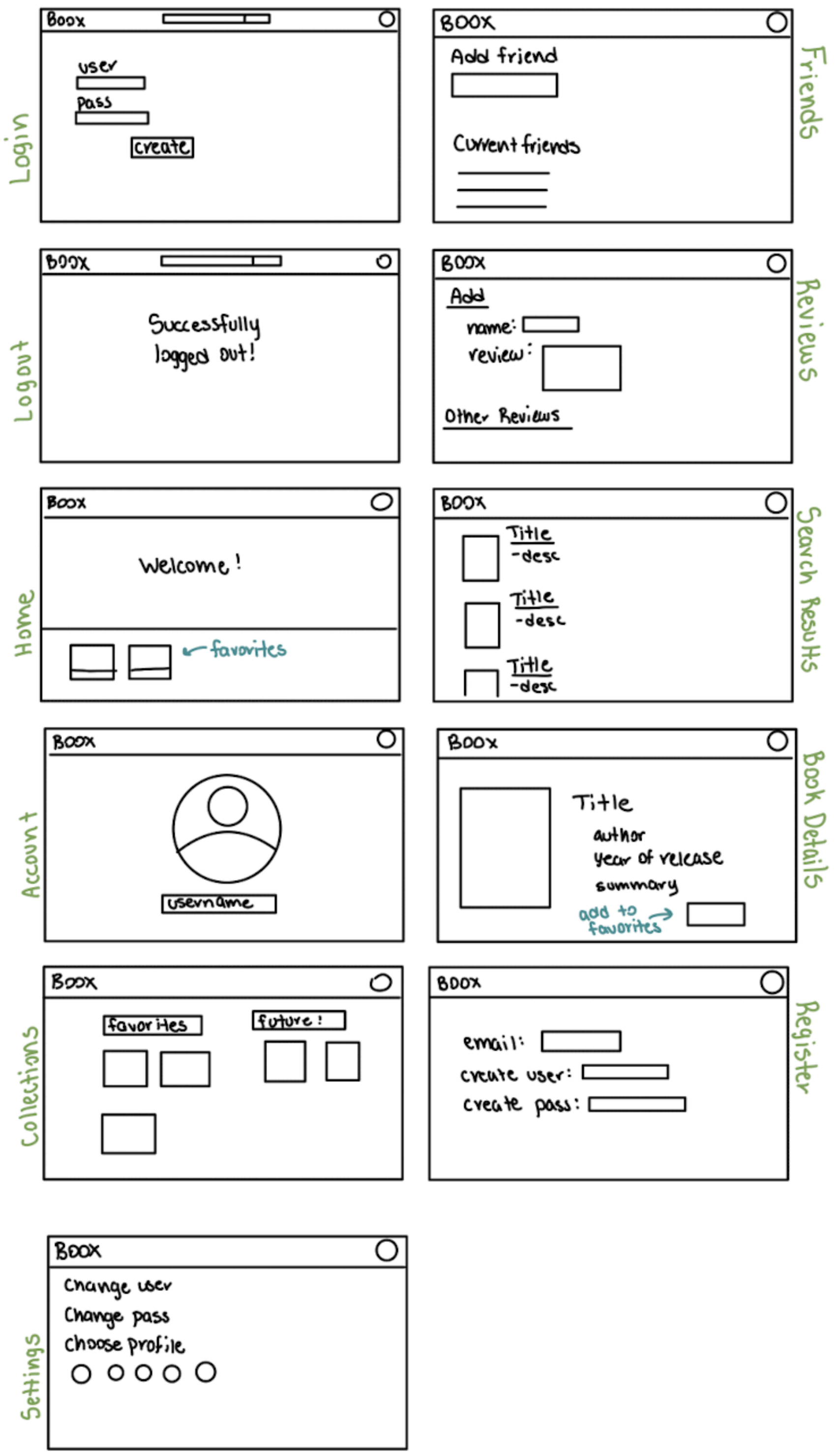
We tasked a friend of the group to assign a book to their favorite collection. We figured this would be a good task as it requires the user to log in, and then search for and add a book. This process tests both the functionality of the website as well as any user error.

- The tester for our project first tried to add a book to their favorites without logging in. Opting to just use the search function from the navigation bar on the login screen, they found their way to the search results. After clicking on a book, it broke the website. This is due to a function that tries to receive the user id from the current session, which wouldn't exist, as the user was not logged in. This resulted in us adding some code to the index.js route to check if a user was logged in before rendering the book detail page, and the correlating test case.
- Additionally, there was an incident where the user accidentally created an account that they couldn't log in to. Initially, there was only one password input box, and while registering an account, they mistyped their desired password. Due to not knowing the password, they had to register a new account. For this new account, they chose the same username as the one that they didn't know the password to. In practice, this could result in some bugs in our SQL database if two users tried interacting with the website while sharing the same username. This prompted us to add a confirm password text box and corresponding test case, along with making sure that the username is unique with a test case to go with it.

Use Case Diagram:


User
Login
Change User Settings
Search for Books
Add/Remove Friends
Add Reviews
Text
Make/View Collections
Website
Create/Delete/ Update Users
Display Books, Reviews, Friends, & Collections
Store Books, Reviews, Friends, & Collections

Wire Frames:


Login
Boox
user
pass
create
Friends
BOOX
Add friend
Current friends
Logout
BOOX
Successfully logged out!
Reviews
BOOX
Add
name:
review:
Other Reviews
Home
Boox
Welcome!
favorites
Search Results
BOOX
Title
-desc
Title
-desc
Title
-desc
Account
BOOX
username
Book Details
BOOX
Title
author
year of release
summary
add to favorites
Collections
BOOX
favorites
future!
Register
BOOX
email:
create user:
create pass:
Settings
BOOX
change user
change pass
choose profile

Contributions (Project Board)

Brandon: Found the initial API key and developed the collections page, enabling users to manage their favorite books and future reads. Integrated GET requests to fetch user-specific data from the database, allowing users to add or remove books from their collections. Designed the front end using HTML and Handlebars, ensuring dynamic and responsive display of book collections. Created the necessary routes for interacting with the back-end to update and retrieve collection data. Implemented functionality to update and delete books from collections, providing users with control over their lists and simplifying book management.

Ben: Developed the settings page, allowing users to manage account details such as username, password, and profile picture. Created POST requests to handle changes to user data, including updating the username, changing passwords with hashing, and updating the profile picture. Integrated functionality to reset the profile picture to a default setting and provided feedback to users after each action. Utilized sessions for secure user authentication and ensured that all database updates were executed properly. Designed the settings page using HTML and Handlebars.

Zeke: Built the friends page with functionality for adding and removing friends. Developed GET and POST requests to interact with the database for retrieving and managing user relationships. Integrated dynamic data updates to display friends' reviews, allowing users to interact with their friends' content. Used sessions to ensure secure handling of user data and the SQL database to retrieve and manage relationship information. Also, worked on ensuring the proper display of users' reviews on the friends page, allowing for smooth interaction and easy navigation. Applied proper database queries and utilized Handlebars to render the data dynamically.

Neerja: Designed and implemented the search bar functionality, enabling users to search for books and view detailed information. Utilized API keys and GET requests to retrieve search results and display them dynamically on the book info page. Developed two pages: one to display search results based on the user's input (such as title, author, or ISBN), and another to show detailed book data when a user clicks on a specific book. Integrated query execution to fetch and display this relevant book data, using a html and bootstrap layout on the front end to look cohesive with the rest of the website.

Charlie: Developed the reviews page, focusing on functionality for viewing, adding, and commenting on reviews. Created GET and POST requests to handle retrieving and submitting reviews and comments, interacting with the SQL database to manage user-generated content. Designed the reviews.hbs page to dynamically display reviews for books, allowing users to leave feedback and see others' reviews. Contributed to the creation and insertion of data for reviews and comments in the create.sql and insert.sql files with a reviews table and it's connections.

Ayush: Built the login, logout, homepage, and navbar functionality. Designed the navbar as a Handlebars partial for consistent navigation across all pages. Implemented login and registration forms, handling cases where passwords don't match or users don't exist by prompting the user to re-enter the details or redirecting to the registration page. Developed error handling to ensure proper feedback for incorrect login attempts. Also worked on the homepage layout, providing easy access to main features once users are logged in. Integrated the login and registration processes with the front-end.