CSC 648/848 SFSU Spring 2021 Milestone 4

iShareBooks

Team 04

History Table

Milestone	Date
M4V1	5/13/2021
M3V2	04/26/2021
M3V1	04/22/2021
M2V2	04/03/2021
M2V1	04/01/2021
M1V2	03/10/2021
M1V1	03/05/2021

Github:
https://github.com/CSC-648-SFSU/csc648-04-sp21-Team04/tree/master/application/vertical-prototype/frontend

Team Information

Team Member Name	Role	Email
Yangesh KC	Team Lead/Documentation Expert	ykc@mail.sfsu.edu
Danish Siddiqui	FrontEnd Lead	dsiddiqui@mail.sfsu.edu
Zaid Alkhatib	BackEnd Lead	zalkhatib@mail.sfsu.edu
Aryanna Brown	FrontEnd Member	abrown22@mail.sfsu
Mark Jovero	BackEnd Member / Documentation Expert	mjovero@mail.sfsu.edu
Pramod Khatri	FrontEnd Member	pkhatri1@mail.sfsu.edu
Abishek Neralla	BackEnd Member	aneralla@mail.sfsu.edu
Yuhua Wu	FrontEnd Member	ywu23@mail.sfsu.edu

Table of Contents

1) Product Summary

iShareBook is an online book store that allows users to search, buy, trade and sell used books online based on the school's department, subject, title and author. The books are displayed in tabular format and the buyers can buy books using credit card or paypal, whereas sellers can list books for sale. The two main objectives of the website is to allow users to get books they need at minimum cost from what they would pay in the school bookstore or other online bookstore as well as help the users who want to clear their used books.

Students are left with books that are not needed by the end of each class. Students are still on the lookout for low-cost books. For certain people, the only choice is to resell the books at a low price to the BookStore or Amazon, where they can be resold to students at exorbitant prices. Other students, meanwhile, are on the lookout for some used books. They also pay even higher rates in bookstores. Consider what it would be like if a student could sell their books and set their own prices, as well as purchase textbooks from other students at a reasonable price.

Through our mobile app , iSharebook allows the user to share books with a wide range of people from all walks of life. With iSharebook, a person can list books that they no longer need and set a fair price for them; other people searching for books can visit the web app and, if they locate the book they want, they can contact the vendor and complete purchases. The customer will benefit from not having to spend a substantial sum of money on the books, whilst the seller will get a fair price for the book he or she no longer requires. “Share Book Share Knowledge! ” is our mantra, and it is one that we adhere to. We think it is beyond time to democratize the way we learn. Students may use our website to BUY, SELL, or TRADE their used books and earn money in the process.

For our foreseeable future we plan to make our mobile app service available to the San Francisco State University population. We're actively trying to extend our service to other areas of the country, and we hope to have connectivity to the rest of the world in the near future. Let us all work together to make books affordable to everyone and to make learning fun.

Unique Feature:

Rating system in a book sharing platform is one of the unique features among our competitors. Our rating system allows buyers to rate sellers after they have comfortably bought or traded books from them. This ensures that customers are able to shop with confidence and have an opportunity to rate sellers based on their shopping experience and the deals the seller posted on each book. The rating for each seller is immediately updated whenever a buyer rates them, and it is displayed on their profile. The default rating is set to 5 stars whenever anyone registers an account, but upon posting and selling a book, their rating will be updated by anyone who buys a from them.

URL to deployment server: http://35.215.84.127:3000/\

Name of the product: iShareBooks

P1 Functional Requirements

1. General User:

1.1. A general user shall be able to search books without creating an account.
1.2. A general user shall be able to register for an account.
1.3. A general user shall be able to search for the textbooks available for the trade.
1.4. A general user shall be able to search for the textbooks available for free.
1.5. A general user shall be able to filter searches.
1.6. A general user shall be able to view the privacy policy.
1.7. A general user shall be able to search a textbook with a department.
1.8. A general user shall be able to select the textbooks without creating an account.
1.9. A general user shall be able to view only the inquiries related to the post.
1.10. A general user shall comply with the website terms and conditions.

2. Registered User

2.1. A registered user shall be able to log in to the website account.
2.2. A registered user shall be able to log out from the website account.
2.3. A registered user shall be able to edit his/her account profile.
2.4. A registered user shall be able to list books for sale.
2.5. A registered user shall be able to list books for trade.
2.6. A registered user shall be able to list books for a free giveaway.
2.7. A registered user shall be able to buy books from other sellers.
2.8. A registered user shall be able to trade books with other sellers.
2.9. A registered user shall be able to get books free of cost from other sellers.
2.10. A registered user shall be able to reply to inquiries from the registered users.
2.11. A registered user shall be able to filter searches.
2.12. A registered user shall be able to search books by department names.
2.13. A registered user shall have a rating.
2.14. A registered user shall be able to get a receipt of their transactions.

3. Profile

3.1. A profile shall be created for all registered users.
3.2. A profile shall be created when the account is made.
3.3. A profile rating shall be visible to all registered users.
3.4. A profile shall have a 5 star rating of the registered users by default.
3.5. A profile shall be keeping the records of the textbooks listed.
3.6. A profile shall be added to favorites/save by other registered users.
3.7. A profile's recent transaction shall be visible to other registered user
3.8. A profile shall be able to see their favourites.

4. Payment Type

4.1. A payment type shall be a PayPal account.

5. Textbook

5.1. A textbook shall be posted on sale only by the registered users.
5.2. A textbook shall be accessible with the textbook name.
5.3. A textbook shall be accessible with the department name.
5.4. A textbook shall be posted for trade by the registered user.
5.5. A textbook shall be posted for free by the registered user.
5.6. A textbook shall be purchased by the registered users.
5.7. A textbook shall be publicly available to all registered users.
5.8. A textbook shall be publicly available to all general users.

6. Free Books

6.1. A free textbook shall be listed as free of cost.
6.2. A free textbook shall be visible to registered users.
6.3. A free textbook shall be visible to the general user.
6.4. A free textbook shall have a finalized purchase.

7. Traded Books

7.1. A traded Book shall be available for all general users.
7.2. A traded textbook shall be exchanged between registered users.
7.3. A traded textbook shall be in use or in new condition.
7.4. A traded textbook shall be listed as available for trade.
7.5. A traded textbook's trade shall be finalized.
7.6. A traded textbook shall be removed after the trade is done.
7.7. A traded textbook shall be listed as available for trade.

8. Textbook Image

8.1. A textbook image shall be resized.
8.2. A textbook image shall be the image of a book.
8.3. A textbook image shall belong to each textbook post.
8.4. A textbook image shall be posted by the seller.
8.5. A textbook image shall be clearly visible to all users.
8.6. A textbook image shall be compatible in any device.

9. Receipt

9.1. A receipt shall be made for all purchased textbooks.
9.2. A receipt shall belong to each order of the textbook.
9.3. A receipt shall be uniquely made for an order.
9.4. A receipt shall be a part of the checkout process.

2) Usability Test Plan

Test Plan for Rating System

Test objectives:

The Rating System is being tested. The purpose of this test is to ensure the rating system is easy to use and works properly. A user, upon successful transaction, shall be able to rate the user they transacted with. The rating shall be saved into the database so it can be retrieved. The user should be able to see the correct rating for each user when a profile of another user is clicked. It is also important that a user, upon transaction, can only rate a user once and only once. Lastly, the rating shall be correctly calculated to display the computed average rating of the user. These tests are important because it is integral to how users are moderated. If a user's average rating dips below a threshold, their account shall be disabled.

Test description

System Setup: The system is hosted using G-cloud and is accessible via the world wide web. Server: apache2 -> System Architecture: Ubuntu 20.04

Starting Point: Buyers can rate the seller after they buy a book. The starting point is when the buyer checks out, they get directed to the rating page to rate the seller. The intended users are the sellers and buyers because the seller gets rated and the buyer gives his feedback and rating of the buyer. The tester must already be logged in and has successfully transacted with a seller. This can be achieved by purchasing, trading, or claiming a free book. After the transaction, the tester will see a rating page.

Intended User: Our intended users are mostly students who are in need of an affordable way to purchase course books. The students will have an opportunity to rate fellow students, who are also students, depending on their pricing deals on the books or selling experience with them.

URL of The System: "http://13.58.93.59:3000"

Usability Task description:

Task	Description
Task	Rate a Seller
Machine State	Logged in and successful transaction with seller
Successful Completion Criteria	Rating Updated on Seller's Profile
Benchmark	Completed in 2s

-Usability test table: (test/use case, %completed, errors, comments)

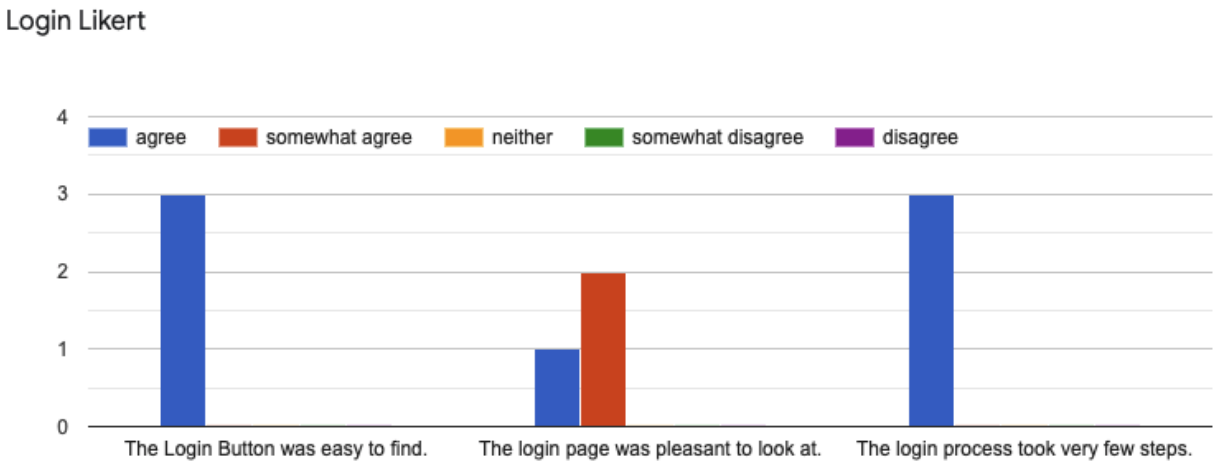
Test	% Complete	Errors	Comments
Rate User	100%	0%	Perfect!
Rating Displayed on Profile	100%	0%	No Errors!
User May Only Rate Once Per Seller	100%	0%	No comment
Rating Saved in Database	100%	0%	Works!
Rating Calculation	100%	0%	Works!

Questionnaire:

The data collected from the questionnaire was taken from all team members and also 5 outsiders. Please note that results may vary for any future survey.

Login Likert

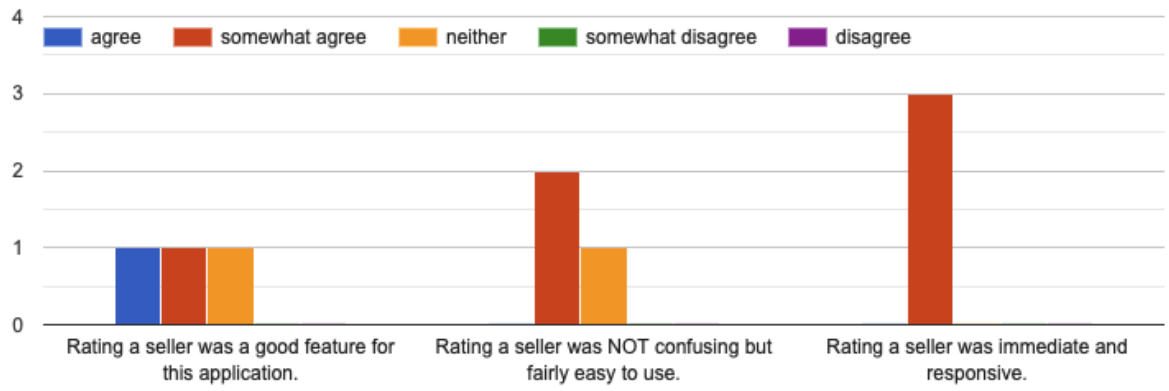
	agree	somewhat agree	neither	somewhat disagree	disagree
The Login Button was easy to find.	○	○	○	○	○
The login page was pleasant to look at.	○	○	○	○	○
The login process took very few steps.	○	○	○	○	○



Rating Likert

	agree	somewhat agree	neither	somewhat disagree	disagree
Rating a seller was a good feature for this application.	○	○	○	○	○
Rating a seller was NOT confusing but fairly easy to use.	○	○	○	○	○
Rating a seller was immediate and responsive.	○	○	○	○	○

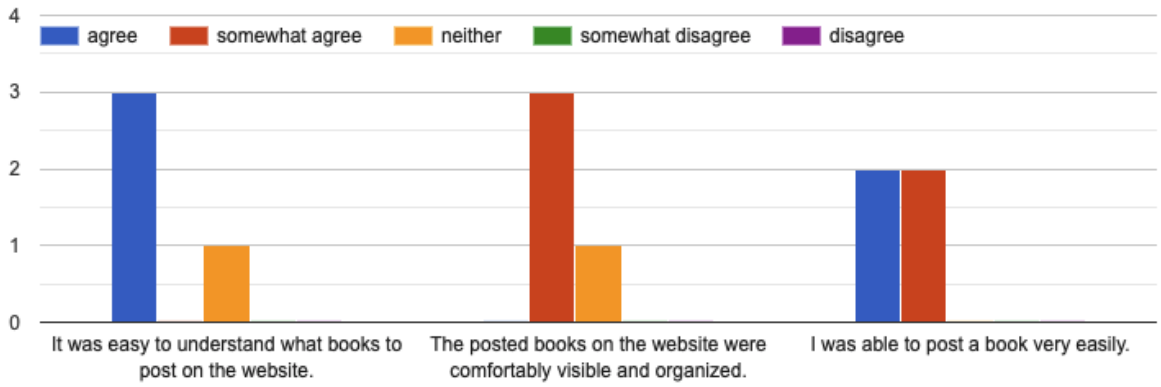
Rating Likert



Posting Likert

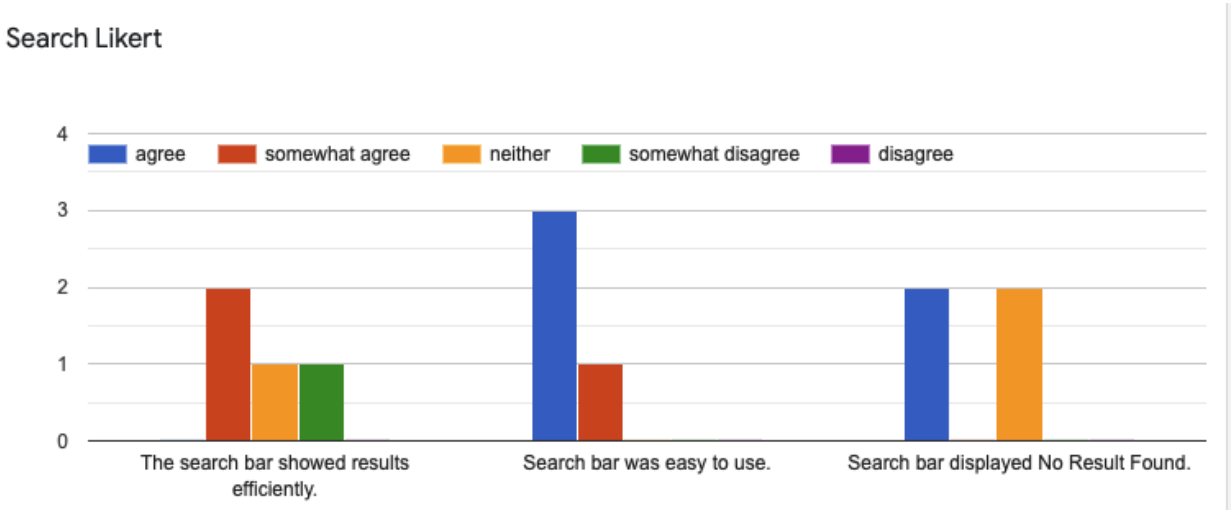
	agree	somewhat agree	neither	somewhat disagree	disagree
It was easy to understand what books to post on the website.	○	○	○	○	○
The posted books on the website were comfortably visible and organized.	○	○	○	○	○
I was able to post a book very easily.	○	○	○	○	○

Posting Likert



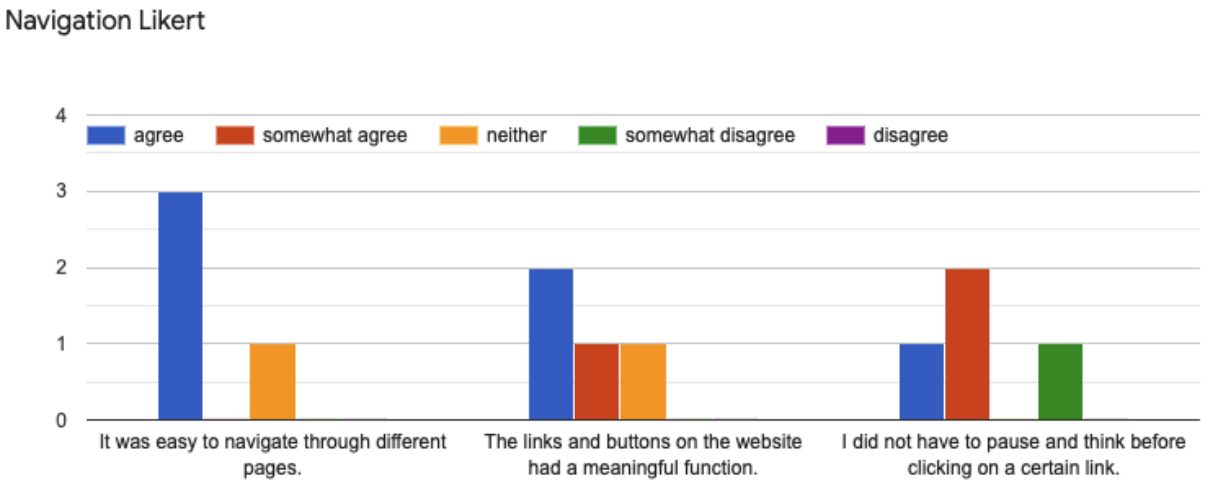
Search Likert

	agree	somewhat agree	neither	somewhat disagree	disagree
The search bar showed results efficiently.	○	○	○	○	○
Search bar was easy to use.	○	○	○	○	○
Search bar displayed No Result Found.	○	○	○	○	○



Navigation Likert

	agree	somewhat agree	neither	somewhat disagree	disagree
It was easy to navigate through different pages.	○	○	○	○	○
The links and buttons on the website had a meaningful function.	○	○	○	○	○
I did not have to pause and think before clicking on a certain link.	○	○	○	○	○



https://docs.google.com/forms/d/e/1FAIpQLSfxSH9ysLGkF6_5_AMpixStslDYWFOHcksGO6KFIX-FhzU8gQ/viewform

3) Quality Assurance Test Plan

1. Test Performance:

Input: Enter **"computer"** in the Search Bar.

Output: Check that you get all the results of the books containing Keyword Computer

2. Password Security:

Input: Enter **"Ykas23123123@!"** with at least 8 characters in the password field.

Output: Check that the password is encrypted in the database to maintain user information privacy.

3. Browser Compatibility:

Input: Enter "http://13.58.93.59:3000" in Chrome, Safari, Firefox, and Edge browsers

Output: Check page loads correctly according to wireframes

4. Prevention of SQL Injection:

Input: Enter **"SELECT * FROM users WHERE id = 3 OR 1 = 1";** in the Login or any input field of the website.

Output: Check that the error "Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://35.215.84.127/search. (Reason: CORS request did not succeed) printed in the console of the browser" is displayed.

5. Memory Leakage:

Input: Inspect the memory section of the Chrome Browser and take multiple memory snapshots.

Output: Check that the memory size of the memory snapshots are stable and equal

6. Test Responsiveness:

Input: Resize the browser screen to iPad and mobile screen sizes.

Output: Check the content of the pages are adjusted according to the screen size.

b) TEST REPORTS

Number	Description	Test Input	Expected Output	Pass/Fail
1	Test Performance	“computer”	Data Loads within 5s without refreshing entire webpage	Pass
2	Test password encryption for Security Checks.	“Ykas23123123 @!”	'$2a$08$mL8V /h6m4i0ZTPT N50YGwO2fn2 kg47x9fzbwf9s mF80TDOCSy 9HLW'	Pass
3	Test Browser Compatibility	Run the web app on Google Chrome, Firefox, and Safari	Web app runs smoothly on all browsers	Pass
4	Test Prevention of Sql Injection Attack	"SELECT * FROM users WHERE id = 3 OR 1 = 1;”	Got an error. SQL Injection Failed.	Pass
5	Test Memory Leakage	window.scroll({ top: 100, left: 100, behavior: 'smooth' });	Warning : memory leaks during the state update on the unmounted component <--------	Fail.

6	**Test responsiveness**	**Run the website on Desktop, Tablet and Phone**	**Responsive for desktop only.**	**Fail**

4) Code Review

Functional Programming.

b) Choose the code (substantial portion of it) related to the features you used for QA and usability tests. You need to submit an example of the code under review (or part of it – 2 pages or so MAX) for this functions to be peer reviewed, and document this as follows:

1. **One team member should submit code to another team member(s) for peer review.**

   Done

   Danish submitted code to Zaid for review, and Zaid thinks it looks good.

2. **Peer review should be performed by other group member(s) (1 review is OK).**

   Done

3. **Peer review is to be done by e-mail and comments are to be included in the code.**

   Done

4. **Submit the e-mail containing the peer review and commented code and email communication related to this in your Milestone 4 document.**

INTERNAL CODE REVIEW:

Danish sent a text file to Zaid via email that includes code portions to be reviewed.

DS

Danish Hassan Siddiqui
Wed 5/5/2021 3:48 PM
To: Zaid Saleh Alkhatib

codeReview.txt
3 KB

Good Afternoon Backend lead,

I wanted to have you review some portions of the code in our app that takes care of the process of rating a seller and display the rating for each user on their profile page.
Attached is a ".txt" file that has the code for you to review and includes both frontend and backend portions.
Any comments and feedbacks are appreciated. You may write your comments on the same file.

Thank You.

Zaid's email response to Danish with his feedback about the code.

ZA

Zaid Saleh Alkhatib
Thu 5/6/2021 10:09 PM
To: Danish Hassan Siddiqui

codeReview.txt
3 KB

Hi Danish,

Thanks for the email. I took a look at the code and overall it looks good. The structure made sense to me and easy to follow, However there were two issues that I think need to be fixed. The first one is the lack of comments, even tho the code looks good but it was hard for me to understand every line of code. The second issue is the console logs(I attached a picture of this issue), this is a production code and I think it shouldn't have console logs everywhere.

Best,
Zaid

```
console.log('RATE');
console.log(ratings);
```

```
console.log(result);
```

Reply | Forward

CODE PORTION OF QA AND USABILITY TEST WITH CODE REVIEW

- **FRONTEND: Logic for rating a user**

```
// Function to stop rating a same user twice

export default function RatingMessage() {                          FEEDBACK:
                                                                   // missing in-line comments
  var confirmation_num = 12314322324;                              // comments are necessary to understand logic
  var name = 'Eden';
  const ratings = useSelector((state) => state.userReducer.ratings);
  const cart = useSelector((state) => state.userReducer.cart);

  var filtered = [];
  console.log('RATE');                                             //console.logs need to be removed for production
  console.log(ratings);

  var filtered_ratings = ratings.filter(function (e, i) {
    if (filtered.includes(ratings[i].name)) {
    } else {
      filtered.push(ratings[i].name);
      return ratings[i];
    }
  });
}

// axis endpoints on frontend to get seller ratings on their profile

React.useEffect(() => {                                            FEEDBACK:
      // fetch data and set seller rating array                    // missing in-line comments
    async function fetchData() {                                   // comments are necessary to understand logic
      const res = await axios.get(
        `http://${window.location.hostname}:3001/get_rating/${sellerid}`
      );
      setSellerRating(res.data.rating);
    }

    fetchData();
  }, []);
```

- **Logic for getting user rating on profile**

```
// axis endpoints on frontend to get user ratings on their profile

 React.useEffect(() => {
      // fetch data and set user rating array                          //good use of useEffect rule
    async function fetchData() {
      const res = await axios.get(
        `http://${window.location.hostname}:3001/get_rating/${userid}`
      );
      setUserRating(res.data.rating);
    }

    fetchData();
  }, []);
```

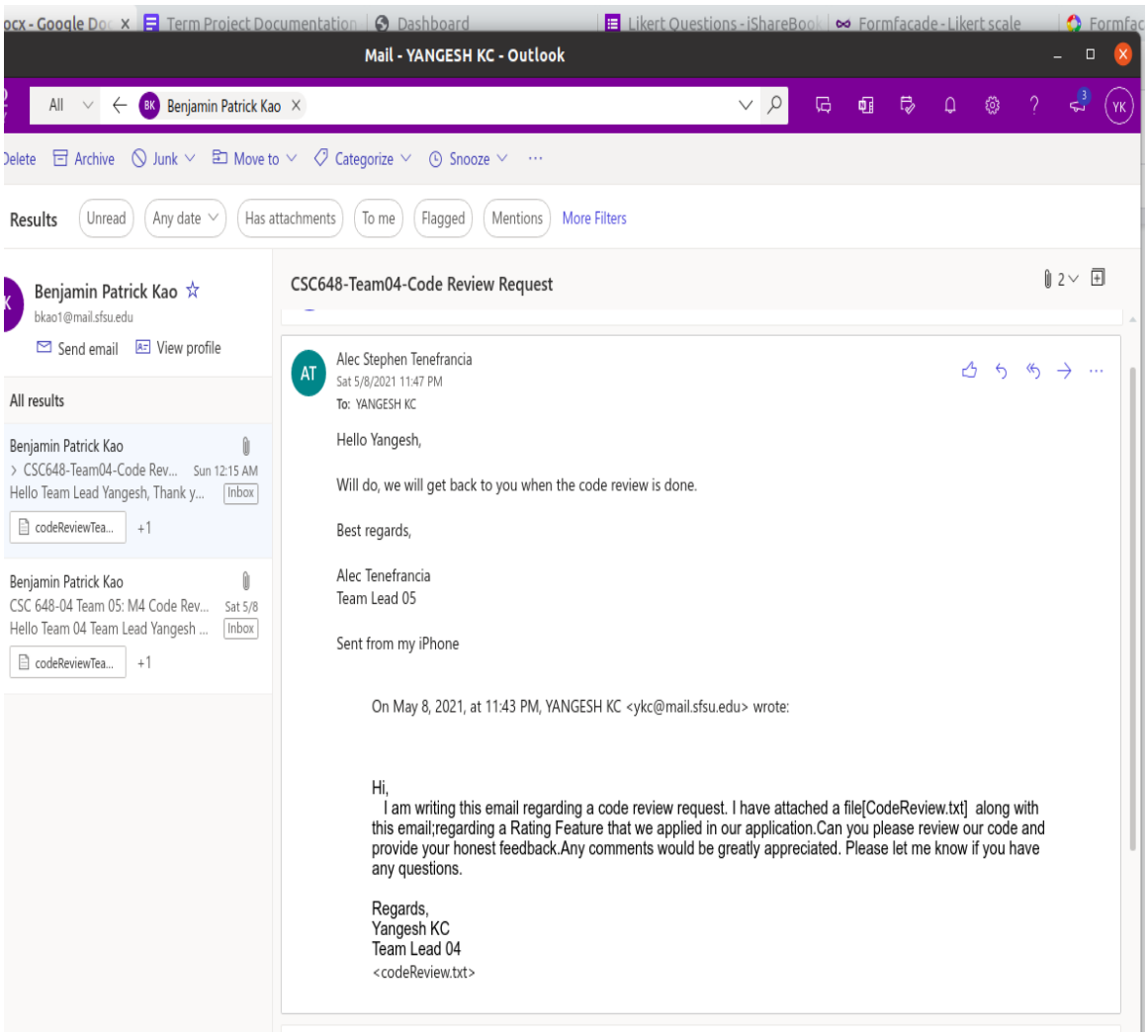
- **BACKEND: Logic for posting a rating to database**

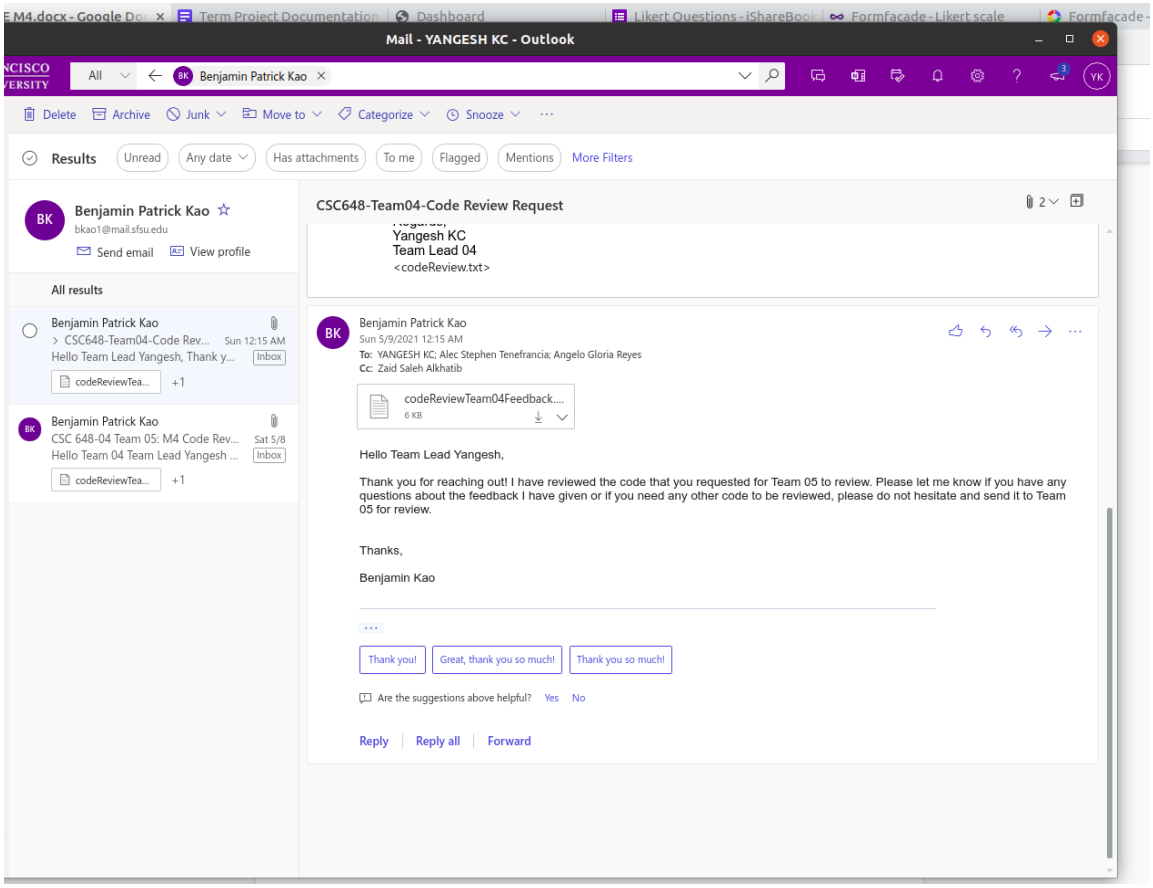
```
// ***************** BACKEND SET UP FOR RATING A SELLER *************

// route to update rating for each seller

router.post('/update_rating/:id', (req, res) => {             //missing in-line comments
  const id = req.params.id;

  console.log(id);                                          //remove console logs
  const newRating = req.body.newRating;
  console.log(newRating);

  let query =
    'UPDATE Ratings SET accumulated_stars=accumulated_stars+' +
    newRating +
    ', total_ratings=total_ratings+1 WHERE user_id=' +
    id;

  db.query(query, (err, result) => {
    if (err) throw err;

    if (result.affectedRows > 0)                                //boolean condition is a good way to identify
      return res.send({                                           correct requests
        succeed: true,
        message: 'Updated user rating.',
      });
    else
      return res.send({
        succeed: false,
        message: 'User does not exist.',
      });
  });
});
```

- **Logic for getting the rating from the database**

```
//route to get rating of a user to display on their profile

router.get('/get_rating/:id', (req, res) => {                    //missing in-line comments
  const id = req.params.id;

  let query = 'SELECT * FROM Ratings WHERE user_id=' + id;

  db.query(query, (err, result) => {
    if (err) throw err;

    console.log(result);

    if (result.length == 0)
      return res.send({
        succeed: false,
      });

    var accum_ratings = result[0].accumulated_stars;
    var total_ratings = result[0].total_ratings;

    return res.send({
      succeed: true,
      rating: Math.round(accum_ratings / total_ratings),          //average rating is nicely done and sent
    });
  });
});
```

EXTERNAL CODE REVIEW - TEAM 05




Mail - YANGESH KC - Outlook
Benjamin Patrick Kao
Delete
Archive
Junk
Move to
Categorize
Snooze
Results
Unread
Any date
Has attachments
To me
Flagged
Mentions
More Filters
Benjamin Patrick Kao
bkao1@mail.sfsu.edu
Send email
View profile
All results
CSC648-Team04-Code Review Request
Yangesh KC
Team Lead 04
<codeReview.txt>
Benjamin Patrick Kao
Sun 5/9/2021 12:15 AM
To: YANGESH KC; Alec Stephen Tenefrancia; Angelo Gloria Reyes
Cc: Zaid Saleh Alkhatib
codeReviewTeam04Feedback....
6 KB
Hello Team Lead Yangesh,
Thank you for reaching out! I have reviewed the code that you requested for Team 05 to review. Please let me know if you have any questions about the feedback I have given or if you need any other code to be reviewed, please do not hesitate and send it to Team 05 for review.
Thanks,
Benjamin Kao
Thank you!
Great, thank you so much!
Thank you so much!
Are the suggestions above helpful? Yes No
Reply
Reply all
Forward

*** M4 Code Review: codeReviewTeam04Feedback.js By Team 05**

*** Peer Reviewed By: Benjamin Kao**

*** Date: 05/08/21**

*** General Comments and Feedback:**

*** - You guys should add a file header that explains what this file contains so that in the future when debugging or scanning through the project, it is easy to just look at the top of every file**

*** to see exactly what is in the file.**

*** - Your functions are very well commented, but I personally think you should add some more inline comments within your function. Below, in the specific feedback, I mark places where I was a little confused.**

*** - One of the main things about this file that confused me is whether this code was fully backend or if there was some frontend rendering in here. For example, you had router functions which**

*** are obviously part of the backend, but you also had React functions, and although I am unfamiliar with React, it seems to me that these React functions are for rendering the frontend.**

*** - I am not sure how React's recommended folder structure is organized, but based off of my knowledge of NodeJs, I would separate the router functions into its own route directory, so these router functions**

*** would be extracted and put into a rate.js file in the route directory. The React functions would be extracted into the modules directory (although this may be wrong, I only know of the modules folder and**

*** the general idea that React frontend rendering stuff is organized there)**

*** - I would also extract all of the database logic into a database connector where all queries are abstracted to and create Object Relational Mappings (ORMs) that are objects that map to a table in your**

*** database. I think in the long run, this would make your product a lot more scalable and readable.**

*** Overall, I really like how concise your backend code is so far. It is very readable even for me who doesn't know a lot about React and you have some very nice organization comments that section off your code.**

*** Specific Comments/Feedback: Inline comments below**

```
/ Function to stop rating a same user twice



export default function RatingMessage() {

 // Is this number the number that will be used in the final product? If this is hardcoded, I personally like to use TODO comments to show that this isn't finished yet.

 var confirmation_num = 12314322324;

 var name = 'Eden';




 // What kind of object is ratings? It looks like you are accessing data within this ratings object to filter, so maybe adding a comment about what type of data structure this is

 // will be very helpful in the future.

 const ratings = useSelector((state) => state.userReducer.ratings);

 const cart = useSelector((state) => state.userReducer.cart);



 var filtered = [];

 console.log('RATE');
```

```
 console.log(ratings);

2

 var filtered_ratings = ratings.filter(function (e, i) {

   if (filtered.includes(ratings[i].name)) {

   } else {

     filtered.push(ratings[i].name);

     return ratings[i];

   }

 });

}



// axis endpoints on frontend to get seller ratings on their profile



React.useEffect(() => {

 // fetch data and set seller rating array

   async function fetchData() {

     const res = await axios.get(

       `http://${window.location.hostname}:3001/get_rating/${sellerid}`

     );

     // What is this function calling/referencing?

     setSellerRating(res.data.rating);

   }
```

```
    fetchData();
  }, []);

  // axis endpoints on frontend to get user ratings on their profile

React.useEffect(() => {
  // fetch data and set user rating array
    async function fetchData() {
      const res = await axios.get(
        `http://${window.location.hostname}:3001/get_rating/${userid}`
      );
      // What is this function calling/referencing?
      setUserRating(res.data.rating);
    }

    fetchData();
  }, []);

// ***************** BACKEND SET UP FOR RATING A SELLER *************

// I think this logic should be extracted into a separate router file.
```

```
// route to update rating for each seller

router.post('/update_rating/:id', (req, res) => {
 const id = req.params.id;

 console.log(id);
 const newRating = req.body.newRating;
 console.log(newRating);


 // I think there is a slightly better way to do string interpolation here in JavaScript, also, this may be vulnerable to SQL injections, but I am no expert in that stuff.
 let query =
   'UPDATE Ratings SET accumulated_stars=accumulated_stars+' +
   newRating +
   ', total_ratings=total_ratings+1 WHERE user_id=' +
   id;

 db.query(query, (err, result) => {
```

```
    if (err) throw err;


    if (result.affectedRows > 0)
      return res.send({
        succeed: true,
        message: 'Updated user rating.',
      });
    else
      return res.send({
        succeed: false,
        message: 'User does not exist.',
      });
  });
});


//route to get rating of a user to display on their profile


router.get('/get_rating/:id', (req, res) => {
  const id = req.params.id;


  let query = 'SELECT * FROM Ratings WHERE user_id=' + id;
```

```
db.query(query, (err, result) => {
  if (err) throw err;

  console.log(result);

  if (result.length == 0)
    return res.send({
      succeed: false,
    });

  var accum_ratings = result[0].accumulated_stars;
  var total_ratings = result[0].total_ratings;

  return res.send({
    succeed: true,
    rating: Math.round(accum_ratings / total_ratings),
  });
});
});
```

5) SELF-CHECK ON BEST PRACTICES FOR SECURITY

- **List major assets you are protecting**
  - **Password:**
    - Password is encrypted in the backend using bcrypt library which uses salt and produces the hash of the password before it is stored in the database.
  - **Payment Information:**
    - We use Paypal API for validating the transactions and protecting payment data.
  - **Profile:**
    - Only the owner of the account can update/delete their profile information.
    - Other users' profiles are read only.
  - **Post:**
    - Users are limited to viewing/trading posts but are not able to edit post information unless the post belongs to them.
    - Users can post books only if they are logged in.
  - **Rating**
    - A buyer is limited to rate each seller in the transaction to only one time.
    - Users can only rate if they are logged in.
  - **Checkout:**
    - Users can only checkout if they are logged in.
    - Users can only checkout if the cart is not empty
  - **Email:**
    - Unique emails for everyone, needs a distinct email while signing up for a new account.
  - **Env file:**
    - The .env file is only contained within the server and inaccessible by external entities.
  - **Preventing SQL Injection:**
    - User input is escaped to ensure unwanted queries are nested within the input.

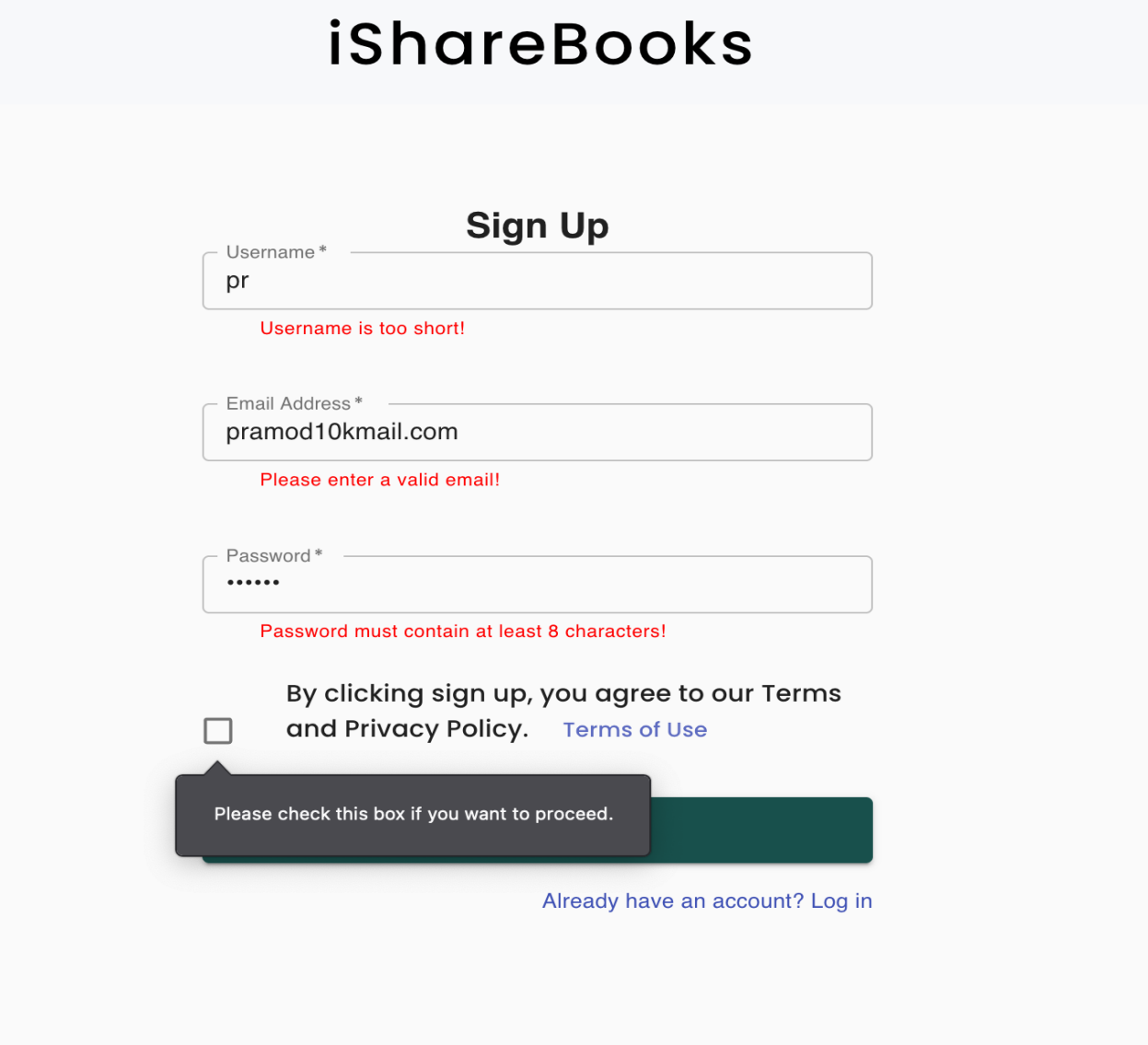
List of Validations:

- Email: The '@' symbol is required followed by the '.' [aka@email.com]

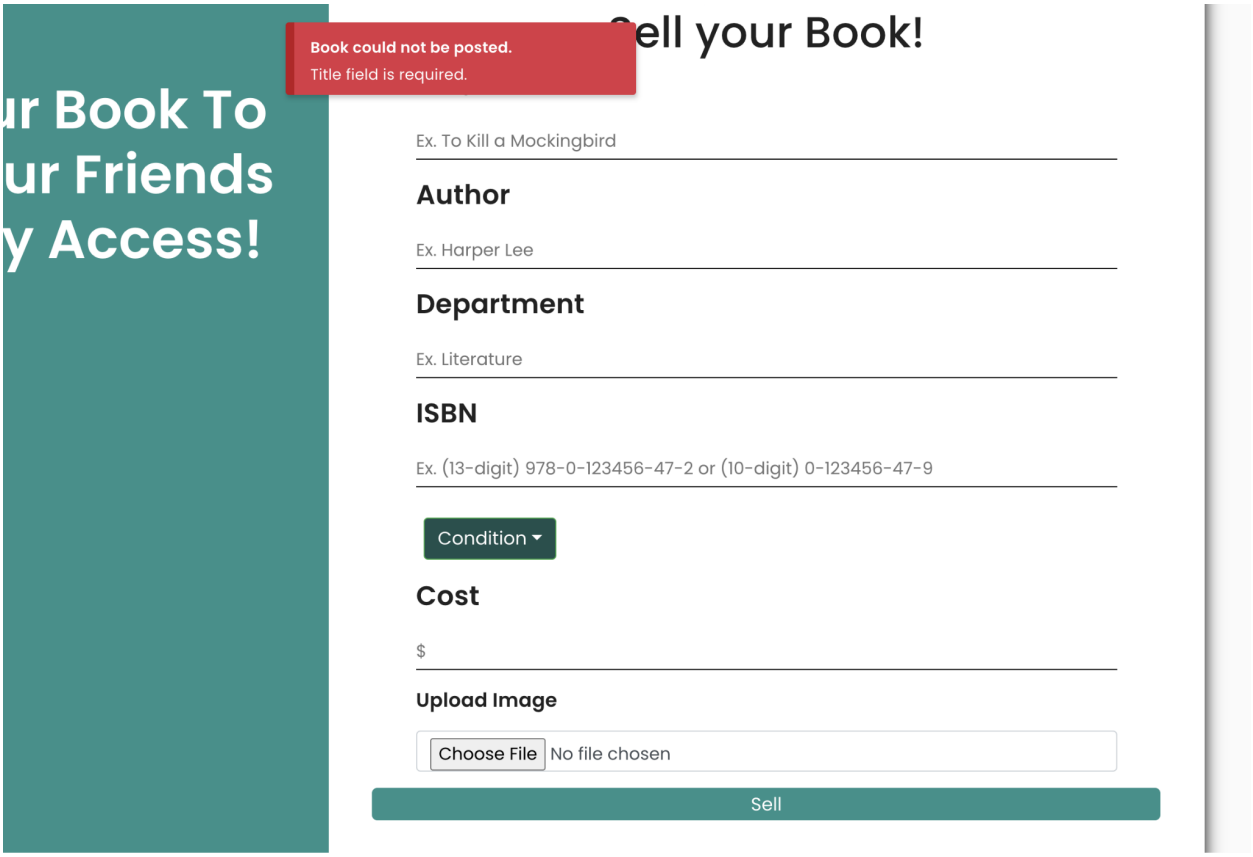
- Password: The password must be at least 8 characters long.

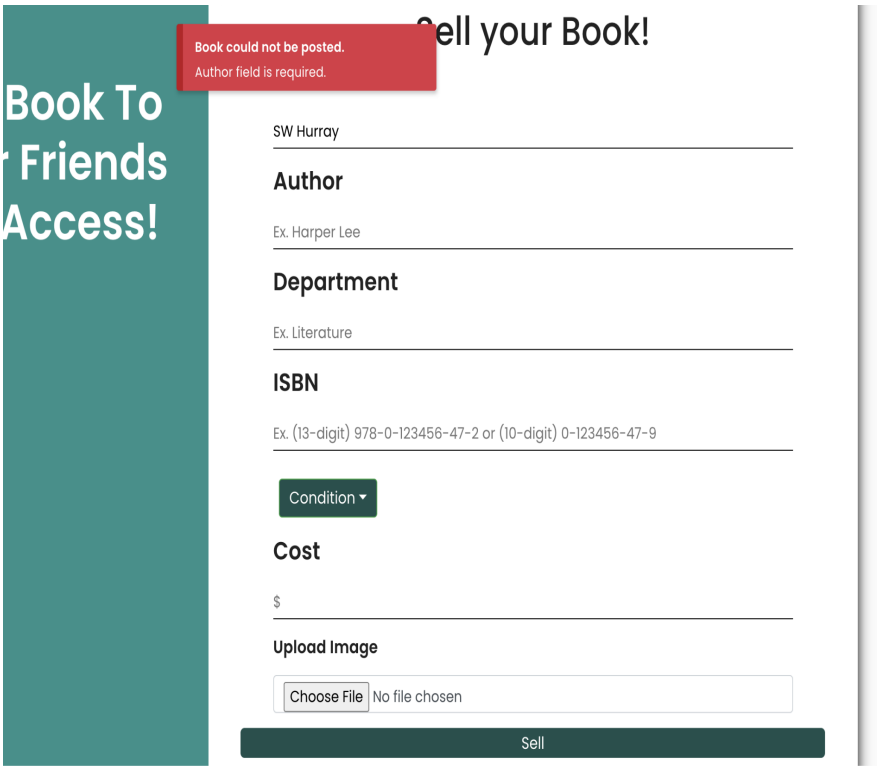
- Username: The username needs to be at least 3 characters long and unique.

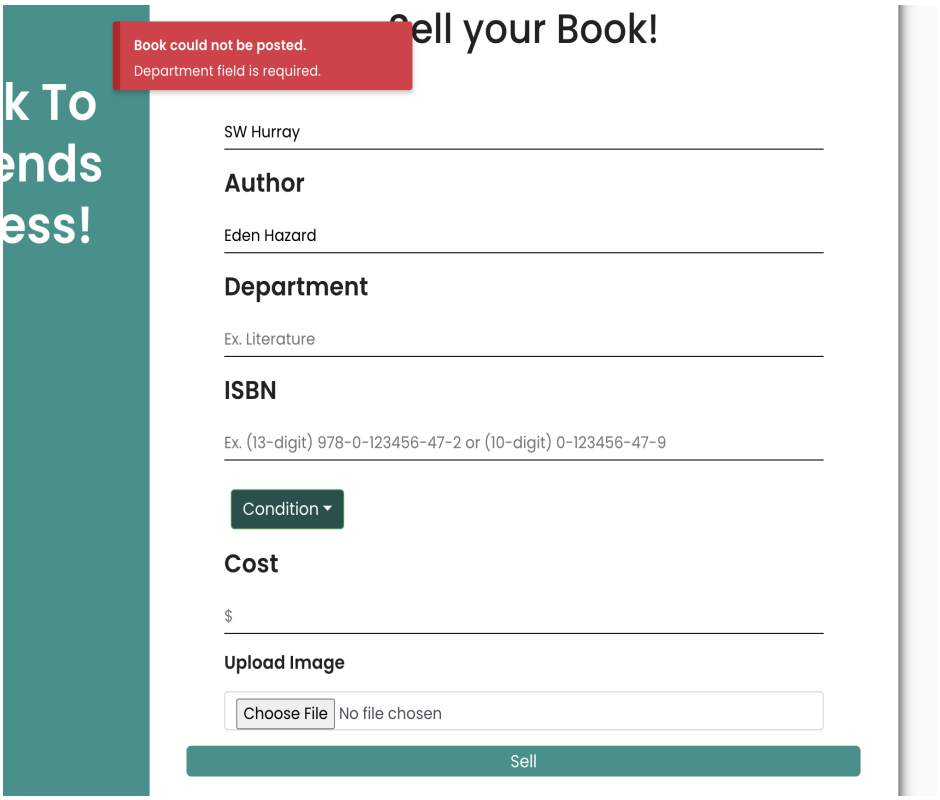
- Checkbox: The checkbox to agree to terms must be checked to register an account.

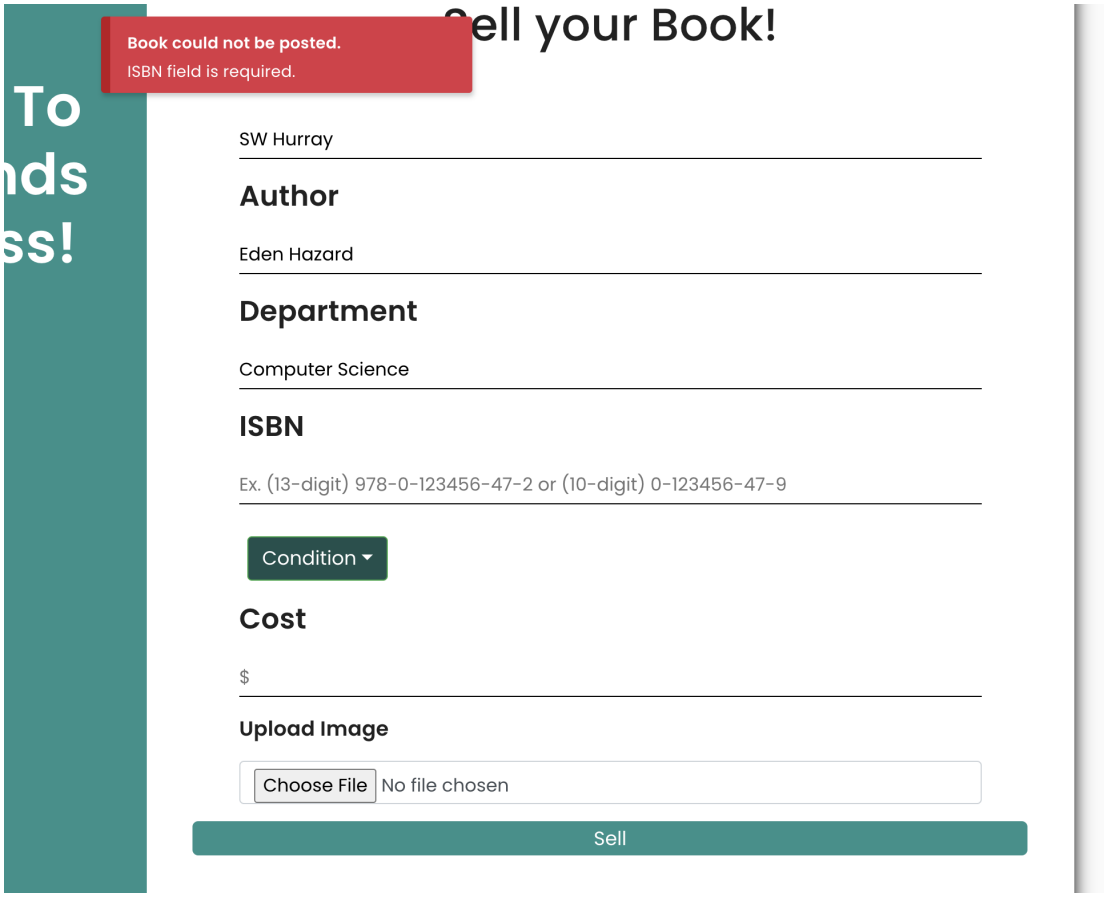


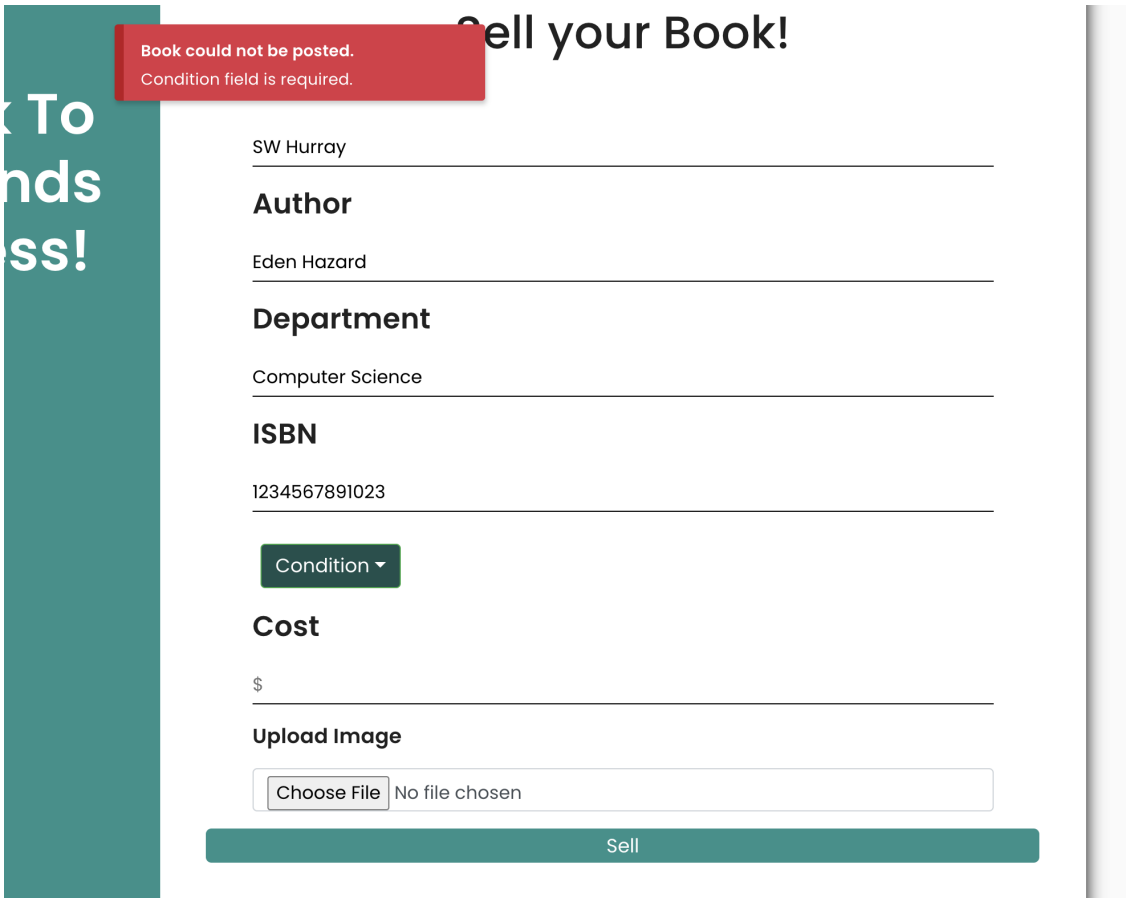
- Title /Author/ Department/ ISBN/ Condition/Cost/Upload Image: None of these fields can be left empty.

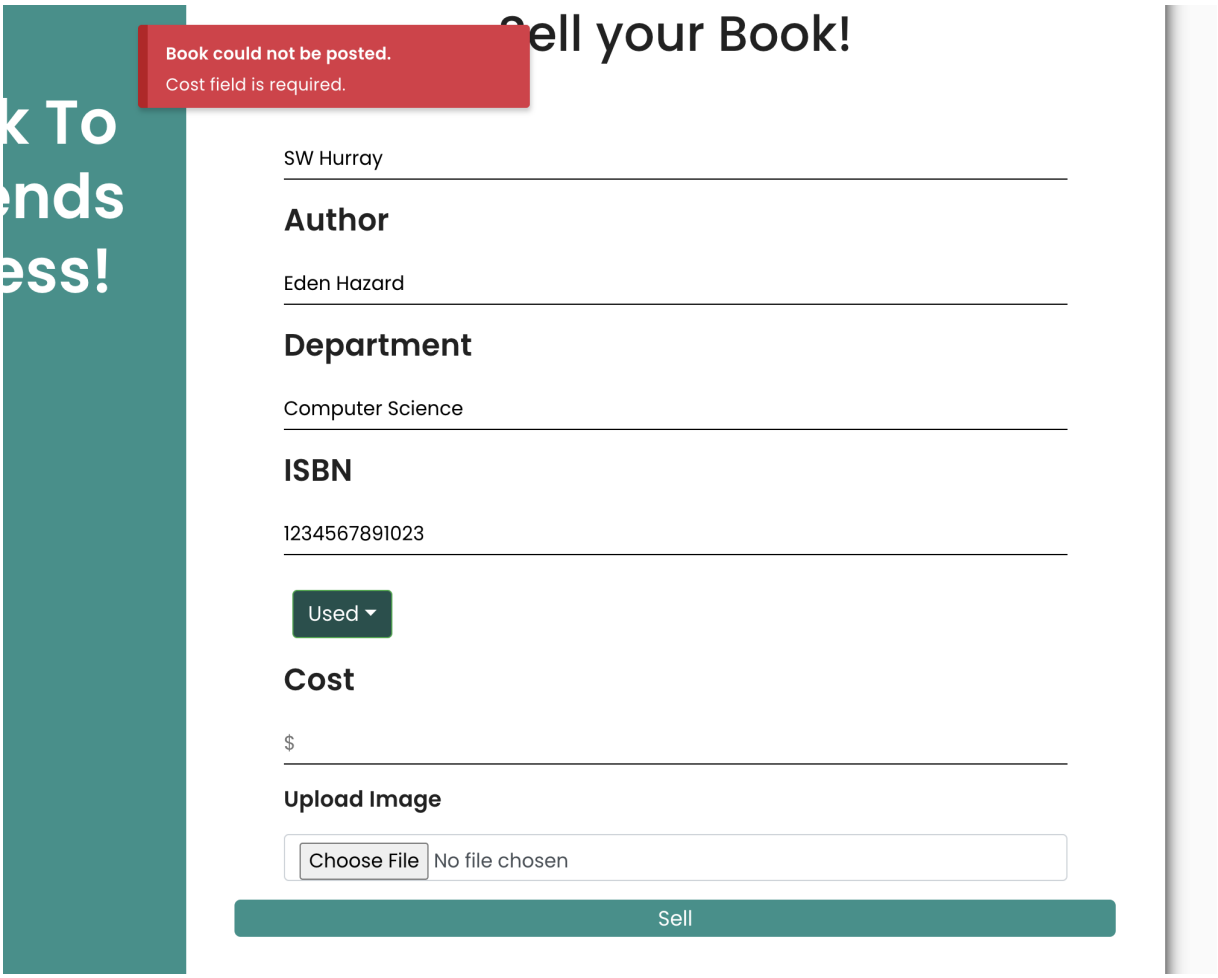


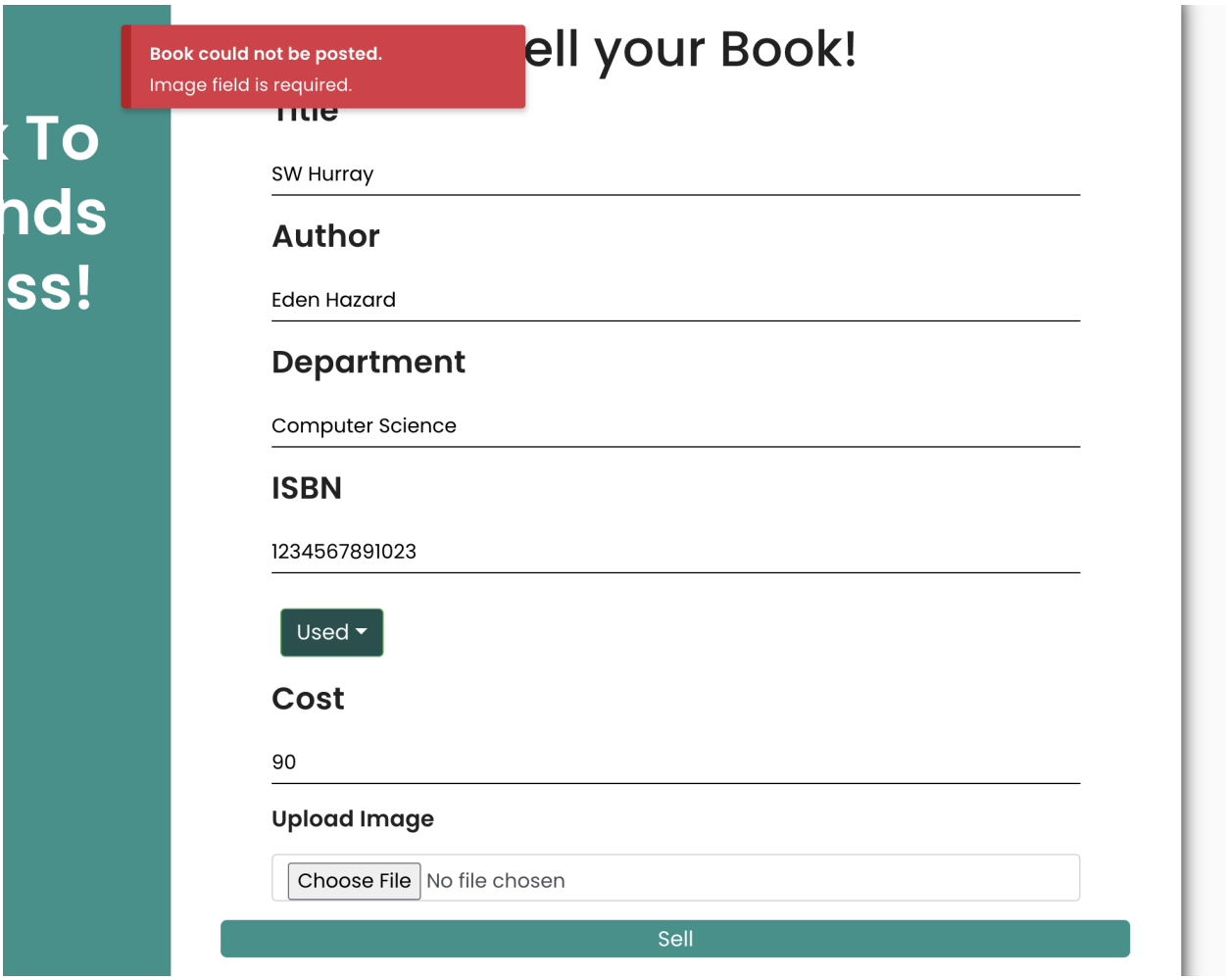

Book To
Friends
Access!
Book could not be posted.
Author field is required.
ell your Book!
SW Hurray
Author
Ex. Harper Lee
Department
Ex. Literature
ISBN
Ex. (13-digit) 978-0-123456-47-2 or (10-digit) 0-123456-47-9
Condition
Cost
$
Upload Image
Choose File
No file chosen
Sell


k To
ends
ess!
Book could not be posted.
Department field is required.
ell your Book!
SW Hurray
Author
Eden Hazard
Department
Ex. Literature
ISBN
Ex. (13-digit) 978-0-123456-47-2 or (10-digit) 0-123456-47-9
Condition
Cost
$
Upload Image
Choose File
No file chosen
Sell


To
nds
ss!
Book could not be posted.
ISBN field is required.
ell your Book!
SW Hurray
Author
Eden Hazard
Department
Computer Science
ISBN
Ex. (13-digit) 978-0-123456-47-2 or (10-digit) 0-123456-47-9
Condition
Cost
$
Upload Image
Choose File
No file chosen
Sell


Book could not be posted.
Condition field is required.
ell your Book!
SW Hurray
Author
Eden Hazard
Department
Computer Science
ISBN
1234567891023
Condition
Cost
$
Upload Image
Choose File
No file chosen
Sell


k To
nds
ess!
Book could not be posted.
Cost field is required.
ell your Book!
SW Hurray
Author
Eden Hazard
Department
Computer Science
ISBN
1234567891023
Used
Cost
$
Upload Image
Choose File
No file chosen
Sell


Book could not be posted.
Image field is required.
ell your Book!
SW Hurray
Author
Eden Hazard
Department
Computer Science
ISBN
1234567891023
Used
Cost
90
Upload Image
Choose File
No file chosen
Sell

```
router.post('/register', (req, res) => {
  const { username, email, password } = req.body;

  var hash = bcrypt.hashSync(password, 8); // Encryption of password, with 8 rounds of salt
  const user = [username, email, hash];

  var insertSQL = `INSERT INTO users (name,email,password) VALUES (?)`;
  db.query(insertSQL, [user], (err, results) => {
    if (err) {
      if (err.sqlMessage.includes('name')) { // Provide the error message to frontend
        return res.send({
          registered: false,
          message: 'Username is already in use!',
        });
      } else if (err.sqlMessage.includes('email')) {
        return res.send({
          registered: false,
          message: 'Email is already in use!',
        });
      }
```

When a user registers, their password will be encrypted using bcrypt. Bcrypt will salt the password with 8 rounds. We also ensure that our queries are protected from injections by escaping the input. With NodeJS, “?” will prevent injections.

Validating Login:

```
db.query(
  'SELECT * FROM users WHERE email = ?', [email], async (err, results) => {
    if (err) {
      res.send(err); // User was not found
    }
    if (
      results[0] === undefined ||
      !bcrypt.compareSync(password, results[0].password) // compare input password with stored password
    ) {
      return res.send({
        auth: false,
        message: 'Password or email is incorrect!',
      });
    } else {  // log in was successful
      const payload = {
        user: {
          id: results[0].email,
        },
      };
      const token = jwt.sign(payload, 'password', { // valid token, allows user to remain logged in
        expiresIn: '90d',
      });
```

Login is validated by using bcrypt to compare the input with the password stored in password.

```
db.query(
  'SELECT * FROM users WHERE email = ?',
  [email],
  async (err, results) => {
    if (err) {
      console.log('ERRRRR' + err);
    }
    if (
      results[0] === undefined ||
      !bcrypt.compareSync(password, results[0].password)
    ) {
      return res.send({
        auth: false,
        message: 'Password or email is incorrect!',
      });
    } else {
      const payload = {
        user: {
          id: results[0].email,
        },
      };
```

Storage of Passwords in MySQL

id	name	email	password	image
85	fight123	fight@gmail.com	$2a$08$WEXmhT0JJRUGKyyeUwcXCOCQT2…	BLOB
86	michael123	michael@gmail.c…	$2a$08$WQhr1wGMBlwQisUscUoUsOl3mE48…	BLOB
87	bierman123	bierman@gmail.…	$2a$08$LyZuVhlkakjy0zYWYVdxEO/2dqtEF9zy…	BLOB

This image shows the encrypted password stored in our database.

Frontend Validation of Input (Registration)

```
// function to validate sign up inputs
const validationSchema = Yup.object().shape({
  username: Yup.string()
    .min(3, 'Username is too short!')
    .required('Username is required!'),
  email: Yup.string()
    .email('Please enter a valid email!')
    .required('Email is required!'),
  password: Yup.string()
    .min(8, 'Password must contain at least 8 characters!')
    .required('Password is required!'),
  university: Yup.string()
    .required('University is required'),
  major: Yup.string()
    .required('Major is required'),
  remember: Yup.boolean().oneOf(
    [true],
    'You must accept terms and conditions.'
  ),
});
```

Each field in the registration (as well as the login) form is validated using the Yup library.

6) SELF-CHECK: ADHERENCE TO ORIGINAL NON-FUNCTIONAL SPECS

Security

No.	Non-Functional Requirements	Status
1.	Verification of email shall be required upon sign in.	**DONE**
2.	Username, password, and email shall be required to create an account.	**DONE**
3.	The passwords shall be saved as encrypted	**DONE**
4.	To post a textbook a user shall be registered with an account.	**DONE**
5.	Photos shall be saved as binary.	**DONE**

Privacy

No.	Non-Functional Requirements	Status
6.	User personal information such as passwords, email, phone number,transactions shall be private.	**DONE**
7.	Registered user's password shall be saved into the MySQL database.	**DONE**
8.	Registered user's email shall be saved into the MySQL database.	**DONE**
9.	Registered user's phone numbers shall be saved into the MySQL database.	**N/A**
10.	Posted textbooks shall be available to all users.	**DONE**
11.	Inquiries about the textbook trade post shall be saved into the MySQL database.	**DONE**

Performance

No.	Non-Functional Requirements	Status
12.	Everytime a user requests something on the website, the requests should be completed within 10 seconds.	**DONE**
13.	The website should load within 5 seconds.	**DONE**
14.	The inquiries between buyer and seller shall be in real time.	**N/A**
15.	The selected textbooks shall stay in the cart upon page refresh.	**DONE**
16.	The website shall be able to display textbooks within 5 seconds upon appropriate search.	**DONE**

System Requirements

No.	Non-Functional Requirements	Status
17.	Technologies implemented in the website shall adhere to the CTO guidelines and implement all the technologies listed.	**DONE**
18.	Website shall support Google chrome.	**DONE**
19.	Website shall support Firefox.	**DONE**
20.	Website shall support Safari and Internet Explorer.	**DONE**
21.	Website shall be simple and user friendly.	**DONE**
22.	Website shall support mobile devices.	**In Progress**
23.	Websites shall be responsive to all devices.	**In Progress**

Marketing

No.	Non-Functional Requirements	Status
24.	Each www page shall display the website logo.	**DONE**
25.	Most www page shall have a contact section at the bottom.	**DONE**
26.	Most www page shall have social media links about the website.	**DONE**

Content

No.	Non-Functional Requirements	Status
27.	A navigation bar shall be present and shall be sticked to the top of the website.	**DONE**
28.	A search bar shall be present within the navbar.	**DONE**
29.	The website logo shall be present to the leftmost on the navigation bar.	**DONE**
30.	The textbooks categorized shall be displayed in the center of the landing page.	**DONE**
31.	The website shall have pleasant colors.	**DONE**
32.	The website shall have trending books listed above textbook categories.	**DONE**
33.	Login and Signup pages shall be accessible from the navbar.	**DONE**
34.	The website shall have separate pages for each textbook category.	**DONE**
35.	Each textbook category page shall have a textbooks posting feature.	**DONE**
36.	Posted textbooks shall have an appropriate image size.	**DONE**
37.	A website footer shall be present in all pages at the bottom.	**DONE**

Functionality

No.	Non-Functional Requirements	Status
38.	The website shall be deployed to Google Cloud.	**DONE**
39.	The website shall be allowing the users to interact in real time.	**DONE**
40.	The website shall be easy to use and easy to navigate.	**DONE**
41.	The website shall have a usage page.	**DONE**
42.	The website shall do efficient and quick searching for the textbooks.	**DONE**
43.	The website shall be allowing users to post many books.	**DONE**

Availability

No.	Non-Functional Requirements	Status
44.	The textbooks shall be available upon appropriate searches.	**DONE**
45.	Upon unsuccessful search, the page should display an error message.	**DONE**
46.	The website should be accessible to any browser.	**DONE**
47.	All the options to buy and sell textbooks shall be visible clearly.	**DONE**
48.	The links to different pages shall be easily tracked.	**DONE**
49.	Checkout should only be accessible when a user is successfully registered.	**DONE**

Fault Tolerance

No.	Non-Functional Requirements	Status
50.	The website shall be able to handle the display errors.	**DONE**
51.	The website shall load all the content upon refresh on an error.	**DONE**

Storage

No.	Non-Functional Requirements	Status
52.	The textbook posts shall be saved into the database without being overloaded.	**DONE**
53.	The textbook posts shall be updated in the database.	**DONE**
54.	The textbook posts shall be deleted from the database.	**ISSUE: Works when a book is sold.**
55.	The general users shall be stored into the database.	**ISSUE: No need because only registered users are saved.**
56.	The registered users shall be stored into the database.	**DONE**
57.	The registered user's account shall be updated into the database.	**DONE**
58.	The registered user's account shall be deleted from the database.	**ISSUE: Not**

		implementing this feature yet, will be included as an update.
59.	The registered user's email shall be saved into the database.	**DONE**
60.	The registered user's password shall be saved into the database.	**DONE**
61.	The registered user's phone number shall be saved into the database.	**ISSUE: Not using phone numbers.**
62.	The textbook's name shall be stored into the database.	**DONE**
63.	Textbook images shall be saved into the database.	**DONE**
64.	Textbook titles shall be saved into the database.	**DONE**
65.	Textbook conditions shall be saved into the database.	**DONE**
66.	Textbook price shall be saved into the database.	**DONE**
67.	All the inquiries about the textbook posts shall be saved into the database.	**DONE**
68.	The department names shall be stored into the database.	**DONE**
69.	The login user's cookie session shall be saved into the database.	**DONE**

Legal

No.	Non-Functional Requirements	Status
70.	The copyright content policy shall be written explicitly at the bottom on all pages.	**DONE**

71.	A link to terms and conditions shall be present at the bottom on all pages.	**DONE**
72.	The website privacy policy shall be clearly visible at the bottom.	**DONE**
73.	The contact information shall be visible to give feedback.	**DONE**

Environmental

No.	Non-Functional Requirements	Status
74.	Final application code shall be sitting in the master branch.	**IN PROGRESS**
75.	Development branch shall act as a mock master branch.	**DONE**
76.	All contributed code shall be done on each member's branch.	**DONE**
77.	The code shall be peer reviewed before merging to the develop branch.	**DONE**
78.	The application shall be run and tested from the develop branch.	**DONE**
79.	The application shall be deployed from the master branch on to Google Cloud Service.	**In Progress**

7) List of Contributions to the Document

1. **Yangesh KC** -***Team Lead***

ykc@mail.sfsu.edu

As a Team Lead:

- *Accountable for following M4 guidelines and ensuring all requirements are met.*
- *Directly supervise team members with daily regular interactions, including managing teams work load.*
- *Managed and oversaw team progress setting up deadlines, group discussion and resolving issues, debugging and testing Product Features.*

As a Team Member:

Milestone 4:

1. *Contributed to the external code review and overall documentation.*
2. *Main contributor to Explore Now Pages and search components.*
3. *Coordinated with team members for code review and quality testing.*
4. *Ensured Security, backup and features of the application are up-to-date.*
5. *Acknowledged code review and worked together with team members to apply necessary feedback.*

\

2. **Danish Siddiqui** ***-Frontend Lead***

dsiddiqui@mail.sfsu.edu

Milestone 4:

1. *Participated in external and Internal Code Review.*
2. *Implemented Frontend logic for comment box only for logged in users.*
3. *Participated in adding screenshots and description for List of Validation.*
4. *Partially completed and finished unique feature description.*
5. *Finished remaining in progress non functional requirements including logged in users credential security.*
6. *Partially participated in writing Quality Assurance testing and Usability Testing description, testing report, and questionnaire.*

3. **Zaid Alkhatib** ***-Backend Lead***

zalkhatib@mail.sfsu.edu

Milestone 4:

1. *Implemented payment using paypal for the backend*
2. *Implemented the login and registration for the backend*
3. *Managed the backend progress with team members*
4. *Helped Fixing the login issue in the frontend*
5. *Helped writing the Q/A tests*
6. *Participated in external and Internal Code Review*

4. **Mark Jovero** *-Backend Member*

mjovero@mail.sfsu.edu

Milestone 4:

1. *Added screenshots for self-check for best practices/security.*
2. *Wrote backend database queries to make sure necessary data is being passed to frontend requests.*
3. *Adjusted ViewBooks page to reflect feedback from CFO.*
4. *Implemented comment box logic such that it is only displayed in books that are traded.*
5. *Validation of post book inputs.*
6. *Contributed to Usability Test Plan.*

5. **Abishek Neralla** *-Backend Member*

aneralla@mail.sfsu.edu

Milestone 4:

1. *Added Frontend Css for the Comment Box and Reviewed Documents for M4.*

6. **Pramod Khatri** *-Frontend Member*

pkhatri@mail.sfsu.edu

Milestone 4:

1. *Participated in self-check adherence to original non functional specs.*
2. *Contributed to self-check on best practices for security.*

7. **Aryanna Brown** *-Frontend Member*

abrown22@mail.sfsu.edu

Milestone 4:

1. *Participation for overall M4 documentation including Self Check, List of Validation, and review of Product Summary.*

8. **Yuhua Wu** *-Frontend Member*

ywu23@mail.sfsu.edu

Milestone 4:

1. *Partial Contribution to peer review/ questionnaire .*
2. *Participation for M4 documentation including Self Check and List of Validation , Reviewed Product Summary.*