Revision #: 1   Date: May 12th, 2019

Software Engineering CSC648/848 Spring 2019

Milestone 4

LitLister

Team 4 - LitLister Team
Vismay Patel - Team Lead, Backend Engineer - vpatel3@mail.sfsu.edu
John Mendoza - Back End lead, Back End Engineer - Jmendo12@mail.sfsu.edu
Jesus Garnica - Front End Lead, Front End Engineer - jgarnica1@mail.sfsu.edu
Edwin Menjivar - GitHub Master, Back End Engineer - emen15@mail.sfsu.edu
Leonid Grekhov - Front End Engineer - lgrekhov@mail.sfsu.edu
Michael Winata - Front End Engineer - mwinata@mail.sfsu.edu

Milestone & Version	Date Submitted For Review	Date Revised
M1 V1	3/14/19	3/19/19
M1 V2	3/19/19	-
M2 V1	4/04/19	4/09/19
M2 V2	4/11/19	-
M3 V1	4/26/19	
M4 V1	5/12/19	

Table of Contents:

1. Product Summary

LitLister

1. San Francisco State University Students and faculty are able to sign up for Lit Lister accounts with their first name, last name, and school email address. After signing up, they will need to verify their email address.
2. Users will only be able to login using their approved credentials once their accounts are verified by email. If the account has not been verified, the user will not be able to log into the website.
3. All users who visit Lit Lister can search for available books using different categories including title, isbn, or author. Registered users will also be able to see all information about posted listings.
4. Registered users who have verified their email will be able to list books for sale, they can specify their own price, the book's condition, and upload their own images of the book.
5. Seller users who have created a listing will have full capacity to edit the price and condition of their book in their listing, as well as deleting their own listings.
6. Seller users shall be able to select a meeting location that is good for their schedule from the predefined on-campus meeting locations, and they will be able to change the location when the buyer user contacts them to agree on a meeting time.
7. Buying users will be able to select a meeting location from the predefined meeting locations that the seller user previously selected. They will be able to select a time slot through messaging, and they can also agree to change the meeting location.
8. Messaging among users will be allowed only after a transaction is started, and the will be able to contact each other to inquire about listings.
9. Users will also be able to rate books and other users to quantify the quality of a book or user.

LitLister is unique, because it connects San Francisco State students and faculty looking to buy and sell books to each other directly. It gives them a platform by which they can create listings of books they would like to sell, and that others in the campus community can buy. Lit Lister also offers users several convenient places on campus where they can meet to exchange their books. Lit Lister powers peer to peer book buying, selling, and trading for San Francisco State. Lit Lister can be found at: https://litlister.com/

2. Usability Test Plan

- Test Objective: For our usability test, the Lit Lister team would like to test book search, listing creation for a specific book, and listing editing. First, we would like to gauge the user's opinions on searching for listings. Second, we want to know our user's feelings about the process of creating a listing for a book. Finally, we would like to examine the usability of editing the price and condition of a listing. We feel these tests are important, because they are the core components of Lit Lister, and they are the basic functions that we would like to provide for our users, so we feel that it is important to ensure that they are easy to use
- Test Description
    - System Setup: Website is deployed on Amazon Web Services running on linux machine. The database is an Amazon RDS. The user will be using at least the fifth latest version of Chrome on a Windows 10 or Unix based machine.
    - Starting Point: The user should start on the *LitLister* homepage without being logged in.
    - Who are the intended users: The intended users are SFSU students who are looking to buy, sell, or browse through books using our website. The test will be performed with SFSU students who are not part of the Computer Science department.
    - URL of the system to be tested: https://litlister.com/
    - What is to be measured: We will measure the user's satisfaction with *Litlister* when searching for listings, creating a listing, and editing a listing.
- Usability Task Description:
    - Task 1: Register & login into the website, and find a listing from our database by performing a search by title.

Task	Description
Task	Search for a listing in website by book title.
Machine state	Home page of *Litlister* https://litlister.com/

	user is logged into an account
Successful completion criteria	User is shown the book and a list of all listings for that given book title.
Benchmark	<=1 minute.

○ Task 2: Login in to the website, and create a listing for one of the books in our database by adding all the necessary information.

Task	Description
Task	Create a listing for one of the books already in the database.
Machine state	Home page of *Litlister* https://litlister.com/; user logged into an account
Successful completion criteria	Check that the user now owns a listing, and that the listing can be seen.
Benchmark	<=5 minutes.

○ Task 3: Login into the website and update the price and condition of a previously created listing by the same user.

Task	Description
Task	Editing the price and condition of a book from your own listing.
Machine state	Home page of *Litlister* https://litlister.com/ , user logged into an account.
Successful completion criteria	Check that the updated information is present and can be seen in the listing.

Benchmark	<=5 minutes.

● Questionnaire:

The listing(s) for the book I searched for were easy to identify & displayed efficiently:
      Strongly Agree   Agree   Neutral   Disagree   Strongly Disagree
      Comments:

The process for searching a listing was very easy and efficient:
      Strongly Agree   Agree   Neutral   Disagree   Strongly Disagree
      Comments:

The process of creating a listing was very simple:
      Strongly Agree   Agree   Neutral   Disagree   Strongly Disagree
      Comments:

It was very easy to edit the price and condition of a listing:
      Strongly Agree   Agree   Neutral   Disagree   Strongly Disagree
      Comments:

It is very easy to find a previously created listing:
      Strongly Agree   Agree   Neutral   Disagree   Strongly Disagree
      Comments:

The website was very consistent:
      Strongly Agree   Agree   Neutral   Disagree   Strongly Disagree
      Comments:

3. QA Test Plan

- **Test Objective:** The objective of this test is to test all aspects of the listing functionality of our website. This will include creating a listing, editing the price and condition of a listing, deleting a listing, searching for a listing, and uploading images to a listing. We will not be testing for features like messaging the creator of a listing, or closing out a listing.
- **Hardware and Software setup:** AWS EC2 instance running Ubuntu with PM2 as the process manager. React.js on the frontend, Node.js on the backend, and MySQL database. We will use Chrome and Firefox as our browsers. The tester will start at the Lit Lister home page already logged into a verified account. **https://litlister.com/**
- **Feature to be tested:** Our objective is to test creating, deleting, searching, and editing a listing; as well as to test image upload for a listing.

QA Test Plan

Test #	Test Title	Test Description	Test Input	Expected correct output	Test Results
1	Listing Creation	A registered user will create a listing to sell a book	An action to create a listing with the following parameters Book : Introduction to algorithms User: jmendo12@mail.sfsu.edu Price: $40	Check for a newly created listing for the book which shows the user as the creator, in addition to the price. Also check for the listing added to the db with book id, user id, and price.	Partial Pass: The user sees their created listing, the listing is added to the database, but on firefox there's a bug when searching for the book to make a listing for

2	Listing Deletion	A registered user will delete a listing they have posted	Navigate to `/user/82/listing/18` which sends the id of the listing to be deleted.	Check for a message stating the listing was deleted, and the listing removed from the db.	Partial Pass: The listing is deleted, but no message is sent to the user.
3	Listing Editing	A registered user will edit the price and condition for a listing they have poster	1 - Login 2 - Create Listing 3 - Memorize listing id in the url Click to edit the listing, and then enter 25 as the new price.	Check that the price for the listing now says 25; check that the price and updated columns in the db contain the new price and time of update.	Pass: The listing is updated, and user is redirected to the page containing updated listing info.
4	Listing Search	A registered user will search for listings for a book	A user will be logged into the site, and will enter "introduction" in the search bar.	Check that the result includes the book and the listing posted by John Mendoza for that book.	Partial Pass: The listing posted is displayed in the search result, but the seller name is not.
5	Image Upload	A registered user will upload images of their book to their listing	An image of the book "Introduction to algorithms" from the testers computer, and the listing for which the image shall be uploaded.	Check that there is a new folder stored on the server containing the image for that listing; check that the db has a link for the photo for that listing; check that the photo is shown when viewing that listing.	Failed: Images are uploaded on user browser, but they are not uploaded onto the database server.

4. Code Review

Our coding style consists of several styles styles used together. First, we use an object-oriented approach in dealing with the bulk of our data. Listings, books, users, and other important software pieces are all structured as objects. Second, we also use some functional programming in our code. Javascript makes functional programming easy, and we use aspects of functional programming such as anonymous methods frequently. Finally, we also use a model-view-controller style, where the front end acts as both the model and the view, and the backend is the controller.
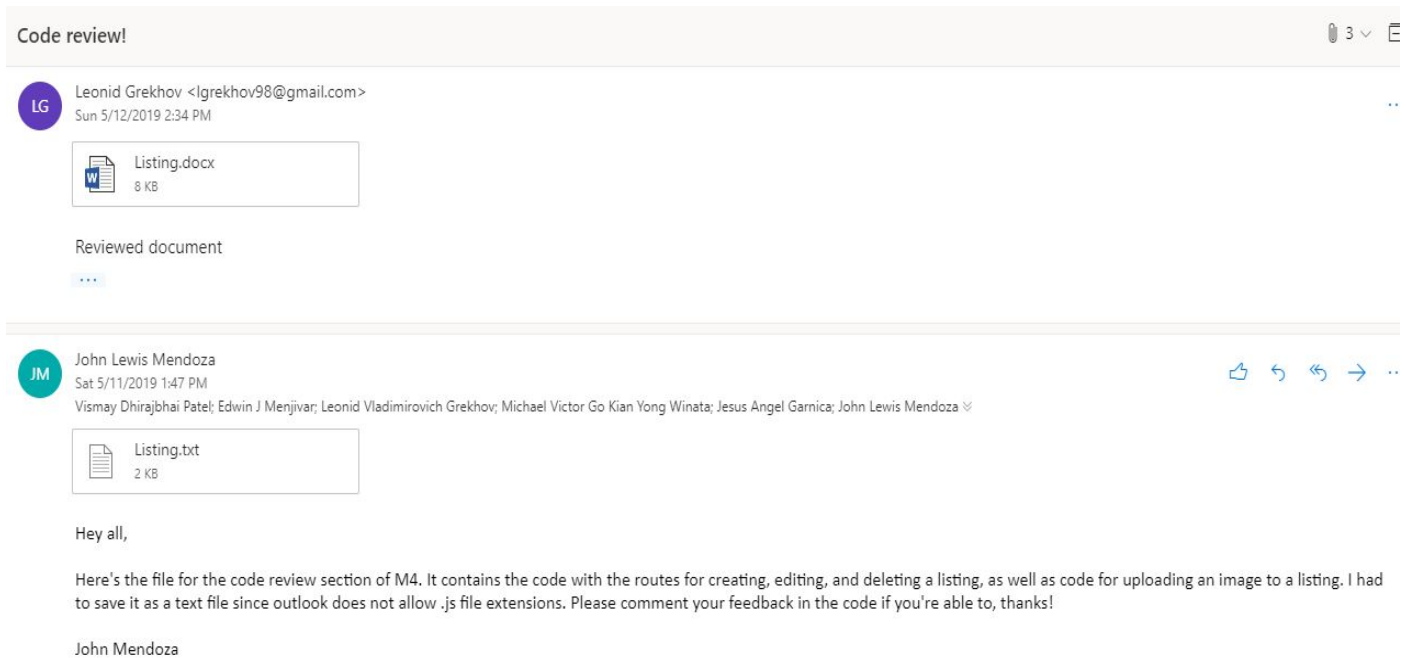

Figure 1. This screenshot shows our code review conducted through e-mail.

```
// Code Review by Leonid Grekhov
// Due to the coding style all functions are self explanatory and don't need comments
// to explain what each function does. Comments can be used in this case to signal visual
// queues to show when a new function begins

// needs header
const router = require('express').Router();
const { Listing } = require('../database/api');
const fs = require('fs');
// function tittle
const imageFileBuffer = fs.readFileSync(
 '/Users/vismaypatel/Desktop/Software Engineering/TheBookProject/csc648-sp19-
team244/database/api/2.png'
);
// function tittle
router.post('/api/listing/create', (request, response) => {
 const { book, user, price } = request.body;
 return Listing.insertListing(book, user, price)
  .then(Listing => {
   console.log(Listing);
   response.json(Listing);
  })
  .catch(error => {
   console.log(error);
   response.json(error);
  });
});
// function tittle
router.post('/api/listing/edit/price', (request, response) => {
 const listingId = request.body.lid;
 const price = request.body.price;

 if (typeof price != 'number') throw 'This is not a valid price';

 return Listing.editPrice(listingId, price)
  .then(Listing => {
   response.json(Listing);
  })
  .catch(error => {
   console.log(error);
   response.json(error);
  });
});
// function tittle
```

```
router.post('/api/listing/edit/condition', (request, response) => {
 const listingId = request.body.lid;
 const condition = request.body.condition;

 if (typeof condition != 'string') throw 'This is not a valid condition';
 if (condition.length > 45) throw 'This condition is too long';

 return Listing.editCondition(listingId, condition)
  .then(Listing => {
   response.json(Listing);
  })
  .catch(error => {
   response.json(error);
  });
});

//Delete listing request
router.post('/api/listing/delete', (request, response) => {
 const listingId = request.body.lid;
 return Listing.deleteLisitng(listingId)
  .then(Listing => {
   console.log(Listing);
   response.json(Listing);
  })
  .catch(error => {
   console.log(error);
   response.json(error);
  });
});
// function tittle
router.post('/api/listing/uploadimage', (request, response) => {
 const listingId = request.body.lid;
 const filename = request.body.filename;
 const extension = request.body.extension;
 const streamData = request.body.streamData;

 //calling a function with test buffer - imageFileBuffer
 return Listing.uploadListingImage(listingId, streamData, filename, extension)
  .then(data => {
   response.json(data);
  })
  .catch(error => {
   response.json(error);
  });
```

Fig 2. This figure shows the code review comments made by Leo. The code will be
further documented based on his comments to improve readability.

- **Instructor's Review**

   Hi John,

   Here is my code review to your code. Try to sent code review by email like I did in this email for M4. That way, it can be easily copied and pasted into your M4 document.

   const router = require('express').Router();
   const { Listing } = require('../database/api');
   const fs = require('fs');

   const imageFileBuffer = fs.readFileSync(
    '/Users/vismaypatel/Desktop/Software
   Engineering/TheBookProject/csc648-sp19-team244/database/api/2.png'
   );

   // I would put a small description in a comment about what each router does.
    // I know that this end-point creates something, but as an engineer reading your code that

```javascript
// is not part of your team, I need more details about what this end-point exactly creates.
// This also applies to all the routers.
// Recall that informative comments are always good in your code, as long as, they don't explain
// something that is already self explained by your code.
router.post('/api/listing/create', (request, response) => {
  const { book, user, price } = request.body;
  return Listing.insertListing(book, user, price)
    .then(Listing => {
      console.log(Listing);
      response.json(Listing);
    })
    .catch(error => {
      console.log(error);
      response.json(error);
    });
});

router.post('/api/listing/edit/price', (request, response) => {
  const listingId = request.body.lid;
  const price = request.body.price;

  // If you are using this comparation with other types in your code
  // What I'd do in this case, is to create an enumerator to compare types.
  // By enforcing to do this with enumerator, you'll reduce the chance of bugs
  // when someone is trying to make the same comparation and made a mistake.
  // If you don't know how to do that here is a good resourse with good examples:
  // https://www.sohamkamani.com/blog/2017/08/21/enums-in-javascript/
  if (typeof price != 'number') throw 'This is not a valid price';
```

Figure 3 to the right shows our that our code was also reviewed by a member of another group.

SR Shivam Rai
Sun 12-May-19 9:10 PM

Hey Vismay,

The code is clean and components are reusable, there is uniform indentation throughout.

Here are the following improvements suggested.
// Add comments regarding all consts use that you are passing
// use camel case for routes
// describe in brief what you are performing in uploadimage and other routes.
// if the methods are tested, remove console logging.
// use relative paths for images

Regards,
Shivam Rai

...

VP Vismay Dhirajbhai Patel
Sun 12-May-19 8:20 PM
Shivam Rai ⌄

📄 Listing.txt
2 KB

5. Self-check on Best Security Practices

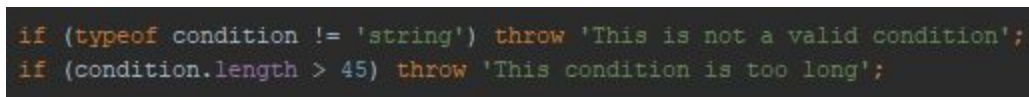Hello Shivam,

Can you perform a code review for a priority 1 functional requirement in our project

...

The major user assets that the Lit Lister team is protecting are our users first names, last names, e-mail addresses, and passwords. All of these aforementioned pieces of data are stored in our database; we keep our database secure with a strong, automatically generated password that we do not share. We keep our user's passwords safe with encryption. We use the bcrypt node package to encrypt our user's passwords before storing them within our database, so we can ensure that our user's account information is kept confidential. We also use passport.js for user authentication; here we use a passport's local strategy to authenticate our users using their username and password. This strategy uses a verify callback to check the credentials, and if they match the user is authenticated. To ensure that our users are real people, we will also use email verification. We also use input validation to ensure that the input we receive from our user's is the input the system expects; if it is not we throw errors that stop the system from receiving bad user inputs.

Fig 4. This screenshot of our database shows that user's passwords are stored as an encrypted hash.

Fig 5. This screenshot shows code that checks user input for editing a book in a listings condition. This code ensures that the user has entered a string with a limit on the length for the condition.

6. Self-check: Adherence to Original Non-functional Specs

Non-functional Specs

1. **Security**
   a. Login shall be required to make purchases - ISSUE payments are not implemented
   b. User's shall verify their emails when registering an account - On track
   c. User's shall be able to set a display name different than their email - On track
   d. User's emails shall not be displayed by default - On track
   e. Passwords shall be encrypted before storing in the database - DONE
   f. User's session timeout limit shall be decided by the administrator - ISSUE no admin backend
   g. User's session shall only be ended by code design - ISSUE when does a session end?
   h. User can login to multiple sessions on different devices - On track
   i. Content uploaded by users shall be audited by the administrator - ISSUE no admin backend
   j. User's payment information shall be encrypted - ISSUE no payment backend
   k. This site shall not accept any third party cookies - ISSUE how can we check this
   l. The meeting time and places users make with each other to exchange books shall not be revealed to other users not involved in the transaction - ISSUE no transaction backend

2. **Audit**
   a. New registrations shall be audited by the administrator - ISSUE no backend for admin
   b. New registrations shall be approved by the administrator - ISSUE no backend for admin
   c. Users shall not be able to login to administrator accounts - ISSUE no backend for admin
   d. New sale listings shall be approved by the administrator - ISSUE no backend for admin

3. **Performance**
   a. The site loading time shall be less than 2 seconds for all screens - DONE
   b. Application shall be able to retrieve information from the database and react in a timely manner. - DONE
   c. The site shall handle requests asynchronously following a REST format - DONE

4. **Capacity**

    a. The total data storage for the site shall not exceed 80% of the server's capacity for this site - DONE

    b. The website shall be capable of handling at least 50 users - On track

    c. The website shall be scalable, so that new features can be added easily - On track

5. **Reliability**

    a. Downtime for maintenance shall be less than 3 hours per month - DONE

    b. Downtime for maintenance shall not affect the site's main functionality - On track

    c. In all cases, users shall be informed of downtime for maintenance, either via an announcement on the main page, or e-mail - On track

6. **Recovery**

    a. In case of a total site failure, the whole site shall be shut down for revision. - DONE

    b. If the site is broken, the mean time to recovery shall not exceed one day. - DONE

    c. User data is the most valuable aspect and priority will be placed on recovering such data in case of total failure. - ISSUE no backend to implement this

7. **Data Integrity**

    a. Database tables shall be backed up weekly - DONE

    b. Administrator shall be able to execute a recovery if needed - ISSUE no backend for admin

    c. Image sizes shall be restricted to at most 1 megabyte - On track

    d. Images shall be uploaded in jpg, jpeg, or png formats - On track

    e. Images will be saved on Amazon's s3 storage server - On track

    f. URLs to image will be stored on the database - On track

8. **Compatibility**

    a. The site shall be compatible with the last version of the Safari browser version 11.1.2 - DONE

    b. The site shall be compatible with the last version of the Firefox browser version 64 - DONE

    c. The site shall be compatible with the last version of the Chrome browser version 73 - DONE

    d. Third party applications shall not be able to modify any content that may affect the site compatibility - ISSUE no backend for this

    e. Content should be able to be ignored by most popular ad-block services. - ISSUE how can we ensure this

    f. The site shall be able to account for any other compatibility issues created as a result of browser updates in the future - On track

    g. The site should be compatible to escalate to new databases - DONE

9. **Conformance with Coding Standards**
   a. Architecture and design standards shall meet all the requirements listed under the High-level system architecture and technologies used section of this document - On track
   b. Design pattern is to be strictly enforced with all aspects of the site. - On track
   c. Appropriate documentation must be created for all code that is individually written for future maintenance. - On track
   d. Production code shall not have any log or output to the console. - ISSUE; production code does log to the console
   e. All errors must not halt the web application without appropriate error handling. - ISSUE some errors may be unhandled
   f. Only working code that meets all code standards shall be submitted to the main branch of the project repository - DONE
   g. Code shall be thoroughly tested and debugged before being considered working code - On track
   h. All internal errors and exceptions encountered when writing or modifying code shall be stored in a log - On track
   i. Any error that can affect the site's functionality shall be reported to the user - On track
   j. Errors shall be handled in a way that does not affect site functionality - On track
   k. The whole production cycle of the site shall be finished at least one week before the delivery date - ISSUE; we may need to shorten the amount of time
   l. The site shall be tested and debugged as a whole product at least one week before the delivery date - ISSUE we may need to shorten this time
   m. The site shall not be launched without all priority one features finished and working - On track
   n. All major changes to the application shall be discussed by the team and communicated to the class CTO. - On track

10. **Look and Feel Standards**
   a. The application and it's layouts shall look professional - DONE
   b. The site shall be simple, so that it is usable to a wide range of users, and all previously mentioned parties - DONE
   c. Targeted users will be the main priority for ensuring usability and readability. - On track
   d. Elements on screen shall meet the compatibility standards of all supported browsers - DONE
   e. Elements on screen shall meet the compatibility standards of all supported browsers on mobile devices - ISSUE mobile squishes elements together
   f. Elements on screen shall be aesthetically pleasing - DONE
   g. The site shall be able to work correctly without mouse interaction - On track
   h. The site shall be able to work correctly without keyboard interaction - On track

      i.   Elements in screen shall be resized automatically without user interaction when being loaded in all the different platforms supported by the site - DONE

      j.   Application's user interface shall make it easy for user to find what they are looking for. - DONE

11. Internalization / Localization Requirements

      a. The default language of the site shall be English - DONE

      b. The site shall only allow SFSU students to register accounts initially - On track

      c. The site shall be scalable to incorporate other schools into the user base - On track

12. Scalability

      a. There shall be a whitelist of accepted school e-mails that can be updated to incorporate other schools - ISSUE no such whitelist exists

      b. The whitelist shall be stored in a file on the server - ISSUE no such list exists

      c. The CPU instance and storage capacity shall be updated to be able to handle a large amount of users if needed - On track

13. Web Site Policies

      a. A link to the policies of this site shall be always visible in all its pages to be accessible by all the parties - DONE

      b. Users payment information shall be kept confidential and secure - ISSUE no backend for payments

      c. The website shall allow users to register an account. - DONE

      d. Email verification shall be implemented upon registration. - DONE

      e. User's shall agree to application's privacy policy before using the product. - DONE