Software Engineering

CSC 648/848 Fall 2021

Project: RoomMe

Section 02

Team 06

Raul-Izaiah Rodriguez - Team lead - rrodriguez2@mail.sfsu.edu

Jay Jaber - Backend engineer/Deployment

Kaung Htun - Backend engineer

Vy Ngo - Frontend engineer

Nael Yun - Frontend engineer

Tanishq Pradhan - Full-stack engineer/Github master

Vandit Malik - Database manager

Milestone 4: Documentation

Date: Nov 9th, 2021

Date Submitted	**Nov 9th, 2021**
Date Revised	**Dec 5th, 2021**

#1: Product summary:

- Name of product - RoomMe

- Final list of priority 1 functionality: functional requirements committed for delivery (will be our grading criteria)
    1. Users shall be able to register for a personal account.
    2. Users shall be able to log into their account.
    14. Users shall be able to contact each other for inquiries via email.
    3. Users shall be able to list one or more units/rooms to be rented.
    6. Users shall be able edit posted listings.
    7. Users shall be able to delete listings.
    8. Users shall be able to search for rooms to rent.
    9. Users shall be able to rent a room alone or with roommates.
    10. Users shall be able to find new roommates. Users shall be able to post and search roommate listings to find others with similar interests.

- What's unique about our product: What is unique about our product is the ability to filter by amenities required in a room and by interest desired in a roommate.

- Url for our website - http://3.22.208.237/

#2: Usability testing plan:

Chosen feature to test: **Upload listing**

Objectives:

The objective of the upload listing shall give the user the ability to upload contents containing a room or about a roommate. This shall help the user to post their desired contents into the page for other users to observe and achieve a potential customer.

Background/setup:

1. Starting point:

On the home page, the user has to login first before he/she wants to upload a room or a roommate profile. If the user does not have an account, he/she can register for a new account. Signup and Signin buttons are located on the top right of the navigation bar. After the user is logged in, the Upload button will appear on the top right of the navigation bar.

2. Test environment:

Google Chrome, Safari, and mobile (with responsive page)

3. Intended users:

The intended users are people who are having rooms for rent and people who are looking for a new place to move in.

4. How to measure user satisfaction:

In order to measure how satisfied the user is about the uploading feature, the user will be asked to fill out a likert questions form and then they will rate their satisfaction on each question.

Task Description:

1. How to measure effectiveness:

- Filling out information for upload listing
- Ability to put room or roommate as a product
- Information regarding roommate and room are provided on the listing page.

2. How to measure efficiency:

- Uploading Roommate : 50 seconds to 1 minute 12seconds
- Uploading Room : 52 seconds to 60 seconds
- Uploading the Room/Roommate onto the home page : 3 seconds to 5 seconds.

Lickert questions:

This form records the user's opinion about the uploading feature. For each question, the user is provided with five rating options: Strongly Agree, Agree, Neutral, Disagree, and Strongly Disagree. After rating, the user can also give feedback at the bottom of the form if he/she wants to.

	Strongly Agree	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Uploading interface is pleasant to use	X				
Uploading form is easy to do			X		

Be able to provide all needed information on the uploading form		X			
The website is user-friendly	X				
Your comments are highly appreciated					
Room information and amenities list was very clear. Website is modern and visually attractive. Option to upload a photo, however, was not obvious.					

#3: QA testing plan:

Test Objectives:

- Testing out uploading listings for rooms to ensure the user will not experience the inconvenience errors when they upload their room to rent.
- Trying out different unsupported file formats for the room picture won't create the listing for the room and give an error.
- Giving incomplete information while creating a room listing doesn’t allow us to submit without providing the necessary information.

Testing environment:

Url for website - http://3.22.208.237/

Safari Version 15.1 (17612.2.9.1.20) both on M1 macbook and intel macbook
Mobile Safari - iPhone
Brave Browser Version 1.31.91 Chromium: 95.0.4638.69 (Official Build) (64-bit)

Feature to be tested:

1. Create a Room listing with everything single details such as correct title, description, price, address, lease-period, room count, occupance, amenities, and possible room images.

2. Create a room listing with mp3.file instead of the correct format for the room picture.

3. Create a room listing with incomplete information such as no address input.

QA Test Plan:

Test Number	Test Title	Description	Test Input	Expected output	PASS/FAIL
1	**“upload room test”**	**Create a new room listing with room picture and description**	**Description Price Address Title Lease-period Room count Occupancy Amenities Image***	**Save the room listing on the database and put the room listing on the webpage for the other user to view**	**Pass for Brave/Safari Browser**

2	**“upload room test”**	**Create a new room listing with mp3 file instead of picture format**	**Wrong format on the room picture. We use mp3 format.**	**Error in uploading and the room listing won’t be created.**	**Fail for both Brave/Safari Browser**
3	**“upload room test”**	**Create a new room listing without the address fill out**	**No address input**	**Ask the user to fill the address before submitting again.**	**Pass for Brave/Safari Browser**

#4: Code review:

Comments: Most of the files at this moment are missing out comments and description of the code written which might lead to confusions in the mind of peer developers to understand some of the challenging logic parts being done at different places.

Some of the code snippet examples are pasted below

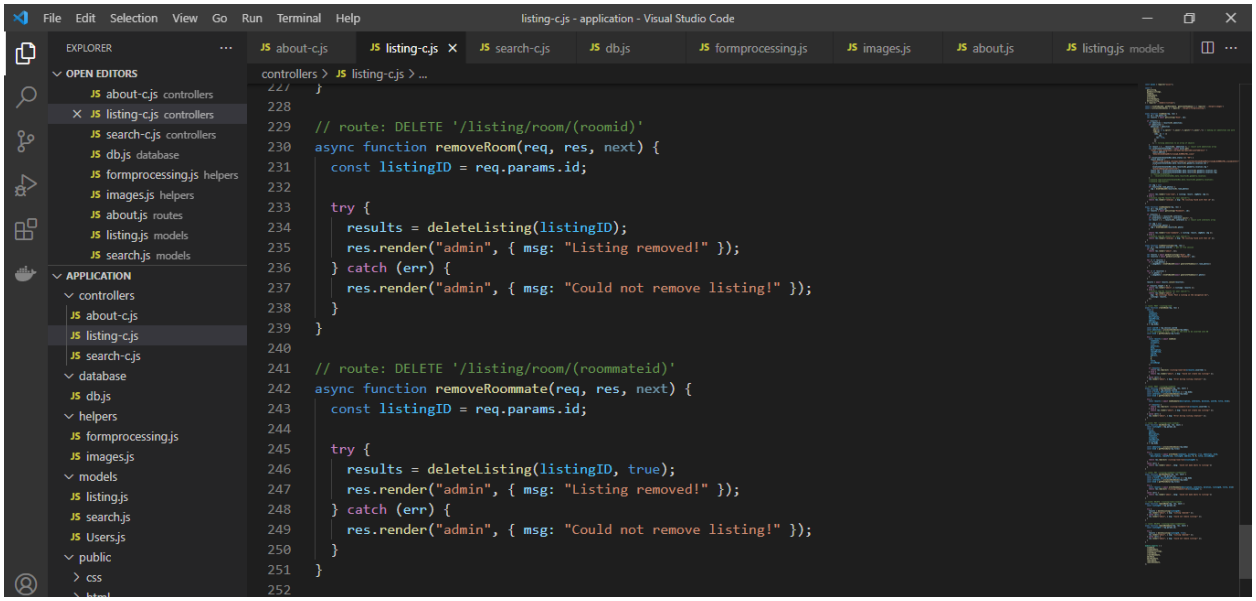
```
const { blobToBase64, generateThumbnail, appendThumbnails } = require('../helpers/images')

async function param (req, res) {
  let type;
  const params = {
    backyard: req.body.backyard === 'on' ? 'backyard': null,
    furnished: req.body.furnished === 'on' ? 'furnished': null,
    gym: req.body.gym === 'on' ? 'gym': null,
    internet: req.body.internet === 'on' ? 'internet': null,
    laundry: req.body.laundry === 'on' ? 'laundry': null,
    parking: req.body.parking === 'on' ? 'parking': null,
    pool: req.body.pool === 'on' ? 'pool': null,
    bathroom: req.body.bathroom === 'on' ? 'bathroom': null,
    utilities: req.body.utilities === 'on' ? 'utilities': null
  }

  const interests = {
    exercise: req.body.exercise === 'on' ? 'exercise': null,
    partying: req.body.partying === 'on' ? 'partying': null,
    food: req.body.food === 'on' ? 'food': null,
    quiet: req.body.quiet === 'on' ? 'quiet': null,
    gaming: req.body.gaming === 'on' ? 'gaming': null,
    reading: req.body.reading === 'on' ? 'reading': null,
    media: req.body.media === 'on' ? 'media': null,
    sports: req.body.sports === 'on' ? 'sports': null,
  }

  const genders = {
```

In this example it might be hard to understand what exactly is done in the param function without comments.

```
const emailExist = (email) => {
    return db.execute("SELECT * FROM User WHERE email=?", [email])
        .then(([results, fields])=>{
            return Promise.resolve(!(results && results.length == 0));
        })
        .catch((err) => Promise.reject(err));
}

const updateUser = (name, password,email, phone_number,gender,id) => {
    return bcrypt.hash(password, 15)
        .then((hashedPassword) => {
            let baseSQL = "UPDATE User SET  name=? , email=? , password=?, phone_number=?, gender=? WHI
            return db.execute(baseSQL,[name, email, hashedPassword,phone_number,gender,id])
        })
        .then(([results, fields])=>{
            if(results && results.affectedRows){
                return Promise.resolve(results.insertId);
            }else{
                return Promise.resolve(-1);
            }

        })
        .catch((err) => Promise.reject(err));
}

module.exports = {
  create, phoneExists, emailExist, updateUser
```

More Examples on Missing out comments

Single Responsibility Functions: It is important to not overwhelm a single function with thousands of lines of code and not provide multiple responsibility to a single function which eliminates the changes of a function from being re-usable and also a source for single point of failure. The current code efficiently follows the best practice to have single responsibility functions and therefore allowing easier refactoring in case of error and make it more re-usable.



Code Snipper showing Single responsibility functions

Code Re-Usability: It is immensely important to write code that can be re-used at multiple places and do not require to re-write the same piece of logic again and again at multiple places. The current coding practice in the project follows the code reusability strategy by having helper functions that can perform some of the common functionality by using a single piece of code being called at different places.

```javascript
// helper function: turns activated checkboxes in req.body into SQL format
function processCheckboxes(body) {
  let amenities = "";
  for (const key in body) {
    if (body[key] === 'on') {
      if (key === 'parking') amenities += `garage/parking,`
      else if (key === 'bathroom') amenities += `private bathroom,`
      else if (key === 'utilities') amenities += `utilities included,`
      else {
        amenities += `${key},`;
      }
    }
  }
  return amenities.slice(0, -1);
}

module.exports = {
  processCheckboxes
}
```

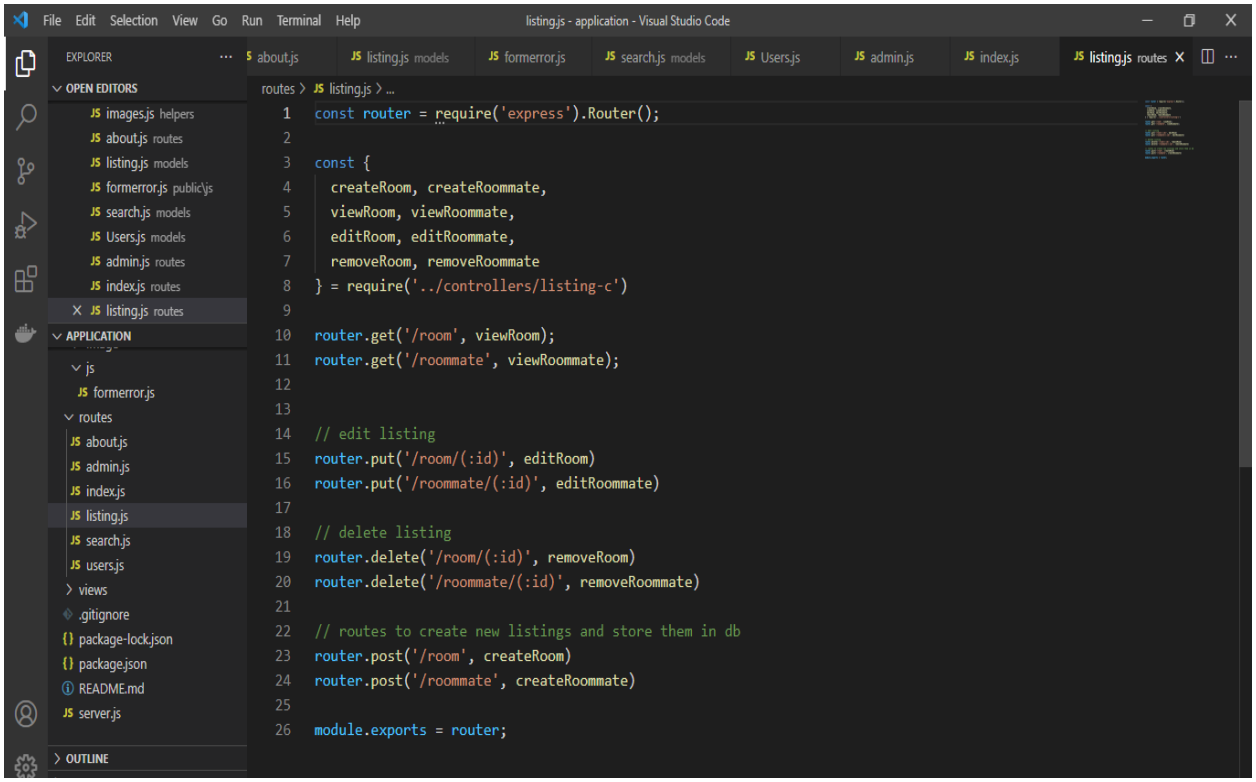
Code Snippet demonstrating helper function that can be used at multiple places in the project

Hardcoding: It is very important to avoid even a single line of code in the whole project that is hardcoded as it increases the chances of making the code more error prone when trying to scale the project since the hardcoded part won't get updated automatically whenever the product is scaled and will have to be manually changed or updated every time new changes are expected in the product. In the current project, there seems to be a little hardcoding in the below code snippet part where the keys are getting checked and the part might require changes as the project starts getting scaled.

```js
// helper function: turns activated checkboxes in req.body into SQL format
function processCheckboxes(body) {
  let amenities = "";
  for (const key in body) {
    if (body[key] === 'on') {
      if (key === 'parking') amenities += `garage/parking,`
      else if (key === 'bathroom') amenities += `private bathroom,`
      else if (key === 'utilities') amenities += `utilities included,`
      else {
        amenities += `${key},`;
      }
    }
  }
  return amenities.slice(0, -1);
}

module.exports = {
  processCheckboxes
}
```

Code Snippet showcasing the keys checked in a hardcoded manner

Scalability: Scalability is an important concept that needs to be raised whenever working on a project as the product can expands quickly and might require big changes without requiring to change the previously written code and hence writing clean code is immensely important which require minimal to no changes during the scaling or expanding the project. Many of the different routes made in the project can be good example of code scalability as they won't require much of a change when the project expands but more and more routes can be added to the project directly.



Code Snippet Example

#5: Self-check on security practices

Assets:

User passwords:

- We protect our passwords by encrypting them before inserting them into the database. That way, even if the database is compromised somehow, the passwords will not be immediately accessible in plain text.

Database access:

- To avoid SQL injection attacks, any form data that is used in DB queries are formatted using a prepared statement. Embedding variables directly in a string allows for unsanitized inputs to be used in DB queries, which is very dangerous.

Javascript runtime (for server):

- To avoid runtime errors crashing the entire server, we need to ensure that our codebase is stable and won't throw any uncaught errors. To that end, any unexpected inputs need to be handled properly, and promises need to be wrapped either in a 'try-catch' or used in '.then' chains.

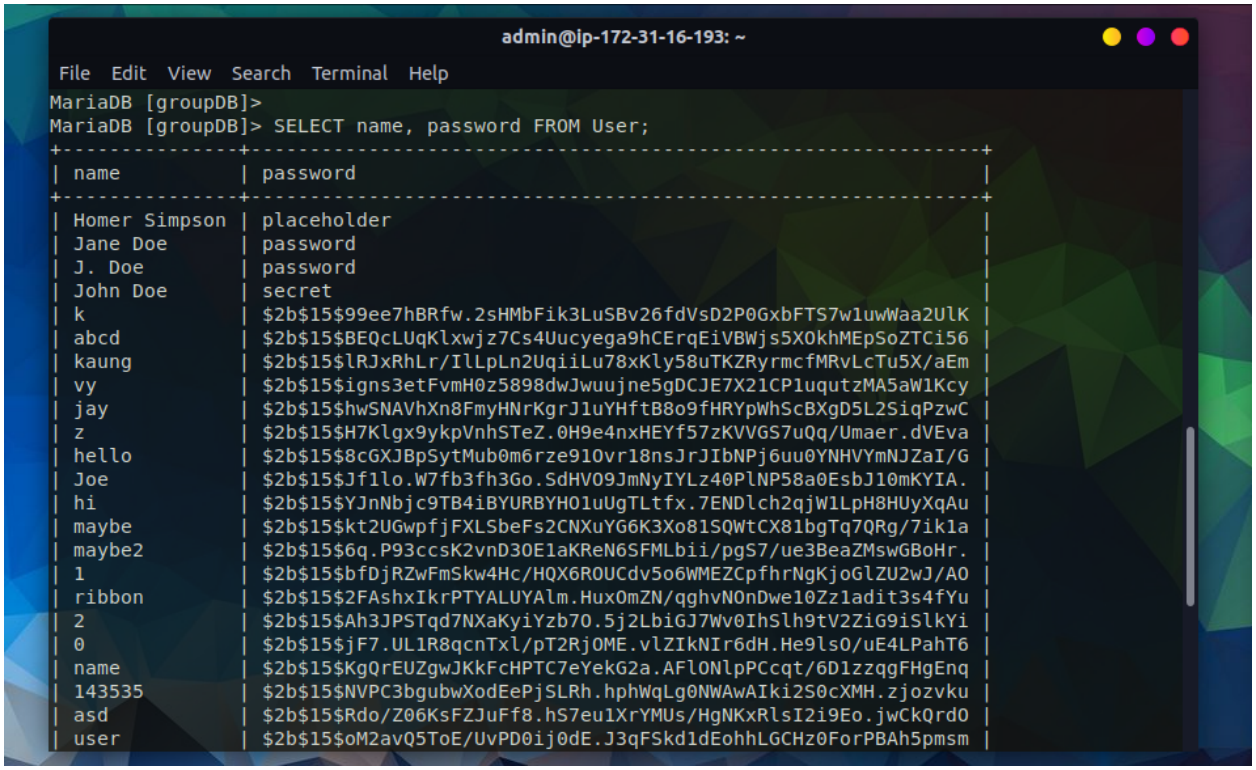
Request/Response cycle:

- If an error occurs somewhere in our route logic, we need to make sure it is communicated to users elegantly instead of the website blocking forever. We added an error page that we forward to when the server errors, and for non-backend errors(bad form input) we will render a message directly in the document.

Form inputs:

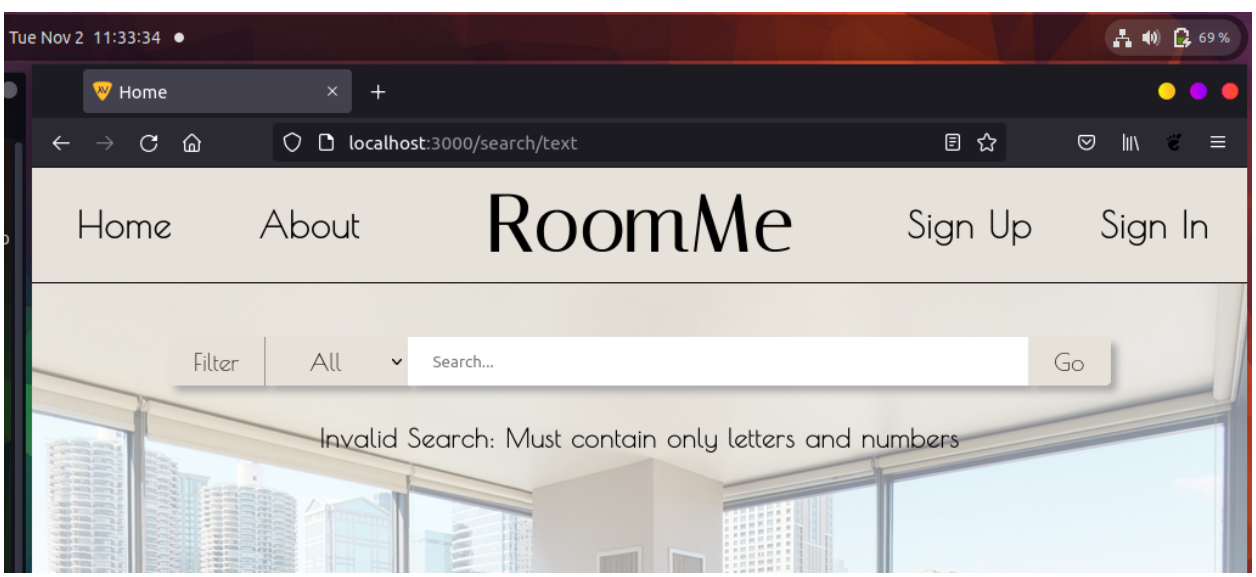
- Inputs provide a mechanism for the outside world to interact directly with the database. Best security practices demand that form inputs be validated on the front and back end, which we are implementing for our site.

Password encryption: first four are from testing before hashing was implemented



```
admin@ip-172-31-16-193: ~
File  Edit  View  Search  Terminal  Help
MariaDB [groupDB]>
MariaDB [groupDB]> SELECT name, password FROM User;
+---------------+-------------------------------------------------------------+
| name          | password                                                    |
+---------------+-------------------------------------------------------------+
| Homer Simpson | placeholder                                                 |
| Jane Doe      | password                                                    |
| J. Doe        | password                                                    |
| John Doe      | secret                                                      |
| k             | $2b$15$99ee7hBRfw.2sHMbFik3LuSBv26fdVsD2P0GxbFTS7w1uwWaa2UlK |
| abcd          | $2b$15$BEQcLUqKlxwjz7Cs4Uucyega9hCErqEiVBWjs5XOkhMEpSoZTCi56 |
| kaung         | $2b$15$lRJxRhLr/IlLpLn2UqiiLu78xKly58uTKZRyrmcfMRvLcTu5X/aEm |
| vy            | $2b$15$igns3etFvmH0z5898dwJwuujne5gDCJE7X21CP1uqutzMA5aW1Kcy |
| jay           | $2b$15$hwSNAVhXn8FmyHNrKgrJ1uYHftB8o9fHRYpWhScBXgD5L2SiqPzwC |
| z             | $2b$15$H7Klgx9ykpVnhSTeZ.0H9e4nxHEYf57zKVVGS7uQq/Umaer.dVEva |
| hello         | $2b$15$8cGXJBpSytMub0m6rze91Ovr18nsJrJIbNPj6uu0YNHVYmNJZaI/G |
| Joe           | $2b$15$Jf1lo.W7fb3fh3Go.SdHVO9JmNyIYLz40PlNP58a0EsbJ10mKYIA. |
| hi            | $2b$15$YJnNbjc9TB4iBYURBYHO1uUgTLtfx.7ENDlch2qjW1LpH8HUyXqAu |
| maybe         | $2b$15$kt2UGwpfjFXLSbeFs2CNXuYG6K3Xo81SQWtCX81bgTq7QRg/7ik1a |
| maybe2        | $2b$15$6q.P93ccsK2vnD3OE1aKReN6SFMLbii/pgS7/ue3BeaZMswGBoHr. |
| 1             | $2b$15$bfDjRZwFmSkw4Hc/HQX6ROUCdv5o6WMEZCpfhrNgKjoGlZU2wJ/AO |
| ribbon        | $2b$15$2FAshxIkrPTYALUYAlm.HuxOmZN/qghvNOnDwe10Zz1adit3s4fYu |
| 2             | $2b$15$Ah3JPSTqd7NXaKyiYzb7O.5j2LbiGJ7Wv0IhSlh9tV2ZiG9iSlkYi |
| 0             | $2b$15$jF7.UL1R8qcnTxl/pT2RjOME.vlZIkNIr6dH.He9lsO/uE4LPahT6 |
| name          | $2b$15$KgQrEUZgwJKkFcHPTC7eYekG2a.AFlONlpPCcqt/6D1zzqgFHgEnq |
| 143535        | $2b$15$NVPC3bgubwXodEePjSLRh.hphWqLg0NWAwAIki2S0cXMH.zjozvku |
| asd           | $2b$15$Rdo/Z06KsFZJuFf8.hS7eu1XrYMUs/HgNKxRlsI2i9Eo.jwCkQrdO |
| user          | $2b$15$oM2avQ5ToE/UvPD0ij0dE.J3qFSkd1dEohhLGCHz0ForPBAh5pmsm |
```

Confirm search bar input validation



#6: Self-check on non-functional specs

- Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0 (some may be provided in the class, some may be chosen by the student team but all tools and servers have to be approved by class CTO).
  ON TRACK
- Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers
  DONE
- Selected application functions must render well on mobile devices
  ON TRACK
- Data shall be stored in the team's chosen database technology on the team's deployment server.
  DONE
- No more than 100 users shall be accessing the app at any given time
  DONE
- Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users.
  ON TRACK
- The language used shall be English.
  DONE
- Application shall be very easy to use and intuitive.
  ON TRACK
- Google maps and analytics shall be added
  ON TRACK
- No e-mail clients shall be allowed. You shall use webmail.
  DONE
- Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.
  DONE
- Site security: basic best practices shall be applied (as covered in the class)
  ON TRACK
- Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development
  ON TRACK
- The website shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Fall 2021. For Demonstration Only" at the top of the WWW page.
  ON TRACK