SW Engineering CSC648/848 Fall 2021

RUMI

Section 02 Team 01

—

Team Lead	➔	Alex Shirazi
(Email: aredjaian@mail.sfsu.edu)		
Github Lead	➔	Jasmine Kilani
Front-end Leads	➔	Nakulan Karthikeyan
		Anmol Burmy
Front-end Team	➔	Jasmine Kilani
		Rasul Imanov
Back-end Leads	➔	Cheng-Yu Chuang
		Joshua Miranda
Back-end Team	➔	Rasul Imanov

—

Milestone 4

(November 9, 2021)

History Table

Revisions	Date
M4 - v1.0	November 9, 2021

Table Of Contents

1 Product Summary

According to an ACS Data study, the city of San Francisco is home to the highest percentage of 18-39 year olds living with roommates. At 28%, the number of young adults living with roommates in the San Francisco Bay Area exceeds that of any other city in the United States. Previously, searching for an apartment to rent or finding a compatible roommate would take countless hours browsing social media housing groups and unorganized forms online. The city of San Francisco and the larger surrounding Bay Area has long called for a reliable solution.

This is why our team has built RUMI- the new age digital housing solution providing a seamless transition into life in the Bay Area. RUMI is a web-based application helping students and young professionals alike find rooms and rent apartments around the San Francisco Bay Area. With RUMI, roommates no longer need to copy/paste their listing specifications countless times in online groups, discussion forms, and insecure sites online. With RUMI, users can enjoy a unique design experience, while making use of plenty of useful features on the platform such as the following:

➔ **Features**:

1. **Guest Users**
   - 1.1. A guest user shall be able to see the available rooms and roommates on the website.
   - 1.2. A guest user shall be able to search any room and roommate available on the website.
   - 1.3. A guest user shall be able to view other users' profiles.
   - 1.4. A guest user shall be able to sign up and make a new account.

2. **Account**
   - 2.1. A guest user shall be able to make a new account.
   - 2.2. An account shall have a unique id for each user.
   - 2.3. An account shall require a username, which must pass the conditions of starting with a character (a-z or A-Z) and having the length of at least 3 characters.
   - 2.4. An account shall require a valid email.

2.5. An account shall require a password, which must pass the conditions of having the length be at least 8 characters long and must contain at least 1 upper case letter and 1 number and 1 of the following special character (/ * - + ! @ # $ ^ &).

2.6. An account shall require the age of the user.

2.7. An account registration shall require the user to accept the Terms of Service and Privacy Rules of the website.

3. Registered User

3.1. A registered user shall be able to login to the website.

3.2. A registered user shall be able to see the available rooms and roommates on the website.

3.3. A registered user shall be able to search any room and roommate available on the website.

3.4. A registered user shall be able to create posts.

3.5. A registered user shall be able to delete their posted post.

3.6. A registered user shall be able to comment on the posts by other users.

3.7. A registered user shall be able to update their profile.

3.8. A registered user shall be able to see other users' profiles.

3.9. A registered user shall be able to copy link, share or save room and roommate’s profile.

3.10. A registered user shall be able to set a preferred search for finding a room (location, price, amenities etc) and roommates (majors, school, smoker?, pets?, language, same interest, hobbies etc).

3.11. A registered user shall be able to get alerts for matching profiles that they are looking for.

4. Profile

4.1. All registered users shall have a profile.

4.2. A profile shall contain an about me and information about the user it belongs to.

4.3. A profile shall contain a profile picture of the user.

4.4. A profile shall contain all the posts posted by the user.

4.5. A profile can be edited by the relevant user.

5. Posts

5.1. A Post shall be posted only by a registered user.
5.2. A Post shall be posted by a user looking for a room.
5.3. A Post shall be posted by a user who wants to rent a room.
5.4. A Post shall contain a photo, a caption or a description.
5.5. A post shall have a post creation date.
5.6. A post shall have a location map for reference.
5.7. A post shall be deleted by the user who posted it.
5.8. A post can be edited by the user who posted it.

6. Comments

6.1. Comments shall be posted by registered users on posts.
6.2. Comments shall contain a comment made on the post and the name of the user who made the comment.
6.3. Comments shall have a creation date.
6.4. Comments shall be deleted by the user who posted it.

7. Messaging

7.1. Messages shall be sent from a registered user to another registered user.
7.2. Messages shall have the message along with the date it was sent on.
7.3. Messages shall be replied to.

8. Search

8.1. Guest users shall be able to search any post on the website
8.2. Registered users shall be able to search any post on the website.
8.3. Search shall be filtered to suit the preferability of the user who is searching.
8.4. Search shall update the contents of the website as the search terms are entered.

9. Admin User

9.1. An admin shall remove any users from the website.

9.2. An admin shall be able to delete posts.
9.3. An admin shall be able to delete comments.

➔ **Live:** http://18.190.48.206:3000

2 Usability Test Plan

➔ Test objectives:

In this usability test, we will be evaluating the post functionality of our website. The post function represents a very important feature that is essentially the backbone of our product and shall be treated with the utmost importance. Its purpose is to be used by our users to publish a room that they have available in order to attract potential users looking for rooms. In a case that this functionality does not work as intended, or has a complicated design/bad user experience, this will create problems for our team and pose a threat for the effectiveness of our product. The main purpose of this test is to reveal any usability or user experience flaws with our post function, evaluate the feedback received from testing, and thus, improve the product as a whole.

➔ Test background and setup:

- System setup:

Our system setup will consist of remote testing. We will be using the video-conference software Zoom and the meeting will be recorded to enable our team to evaluate user behavior and experience closely so that we are able to catch any flaws or problems that the users encountered down to the small details.

- Starting point:

The test will start after participants have joined the meeting. They will be informed of the software setup that is needed to properly carry out the test. They will also be informed that the test will be recorded for future analysis by our team, and they may choose to opt-out if their privacy is a concern.

- Who are the intended users:

There are two general categories of intended users for our product: users seeking a room and users seeking a roommate to share a room that they reside in. For the post feature that this usability test is based upon, the intended users will be either other students or professionals posting a room they have available for potential users seeking a room to reside in.

- URL of the system to be tested :

The usability test will be conducted by visiting the URL: http://18.190.48.206:3000/ and the user satisfaction with the post feature will be measured.

➔ **Usability Task description**:

Please complete the tasks below after successfully creating an account at http://18.190.48.206:3000/ :

Task 1 - Click "Create" on top right hand side of the screen
Task 2 - Upload image of room you want to post
Task 3 - Write a caption for your post
Task 4 - Write a description of what your room offers
Task 5 - Input a price that you would want to receive from potential roommate
Task 6 - Pick the location of your room
Task 7 - Refine filters (parking, pets, smoking allowed?, gender-specificity)

Effectiveness of this usability test would be measured by the results generated from the users following the given tasks to them. In other words, measuring the amount of successful posts by each user would give us an idea of how effective our post function is and if it functions as intended by viewing the quality of the post.

Efficiency can be measured by gathering timing data on completion of the task by each user. We can use the recording of the full duration of a user's experience and depending on the time each user took to complete the task, measure the efficiency of our post function. Essentially, for every user that takes more time than the average of all users to create a post, the less efficient our post functionality is, thus we can improve from there.

➔ **Likert Subjective Test**:

	Strongly Disagree	Disagree	Neither Agree or Disagree	Agree	Strongly Agree
I found it easy to create a post					
I found the information required to create post (picture, caption, description, price, location, filters) effective					
I think the post function needs improvement					

3 QA Test Plan

➔ Test Objectives:

- Our QA test will have an objective regarding the search functionality. Whether the user searches for a roommate or a room, the search algorithm will check what post titles and post descriptions match the user's input text. The search function also counts the number of results that pop up and if 0 results match the user's input, we will notify the user that their search returned 0 results

➔ HW and SW setup:

- Server Host: has been setup on a linux ubuntu server on AWS free tier system with 1vCPU and 1gb RAM
- Browser
  - The site was tested on chrome and firefox
  - The URL for QA testing: http://18.190.48.206:3000/

➔ Features to be tested:

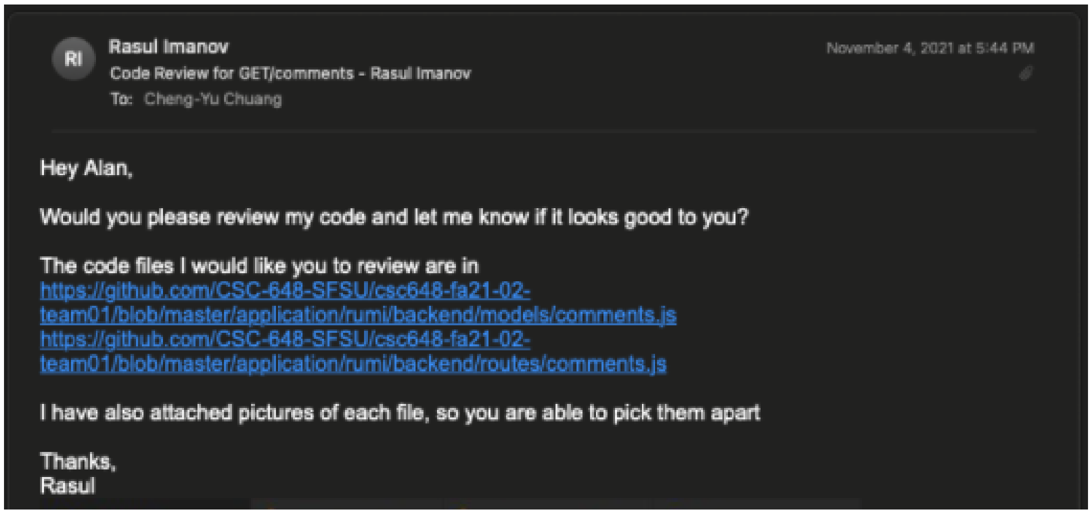
- Filters
  - The user can use provided filters to narrow down search if desired
- Input keywords text search
  - For our search function, the server looks for keywords in post titles and description
- Delete Post
  - If a user owns a post, they can click the link to their listing and delete their post using the delete button
- Register User
  - The user can enter a username and password they wish to make use of, provided the password must pass the conditions of having the length be at least 8 characters long and must contain at least 1 upper case letter and 1 number and 1 of the following special character (/ * - + ! @ # $ ^ &). Afterwards, they can choose to specify their preferences (i.e. what they are looking for, their school, major, etc..) and submit the form. Upon submitting, the user will have a registered profile.
- Login User

  - The user must enter the username and password they specified on registration, which will then allow them to login to the website, allowing them to create posts and logout (relevant icons will pop up where the login and register buttons were on the navbar previously).
- Create Post
  - The user must enter post details (post name, description, price, city, gender, smoking, pets, parking, and a photo of the offer under the create page to make a new post).

➔ QA Test plan:

Test #	Type	Input	Expected Output	Results
1	Filters	Input “sfsu” for school filter	“5 results found”	Chrome:Pass Firefox:Pass
2	Keywords search	Search for “Beach House” in room search	“1 result found”	Chrome: Pass Firefox:Pass
3	Delete Post	Owner hits “delete” button on their post listing	The post is deleted from the user interface.	Chrome: Pass Firefox:Pass
4	Register	A guest user inputs register details to make an account	The guest is able to make an account	Chrome:Pass Firefox:Pass
5	Login	Guest is able to login to an account	The user logs in	Chrome: Pass Firefox:Pass
6	Create	User enters info in post create page to make a listing	“Post successfully created”	Chrome: Pass Firefox:Pass

4 Code Review



RI Rasul Imanov
Code Review for GET/comments - Rasul Imanov
To: Cheng-Yu Chuang

November 4, 2021 at 5:44 PM

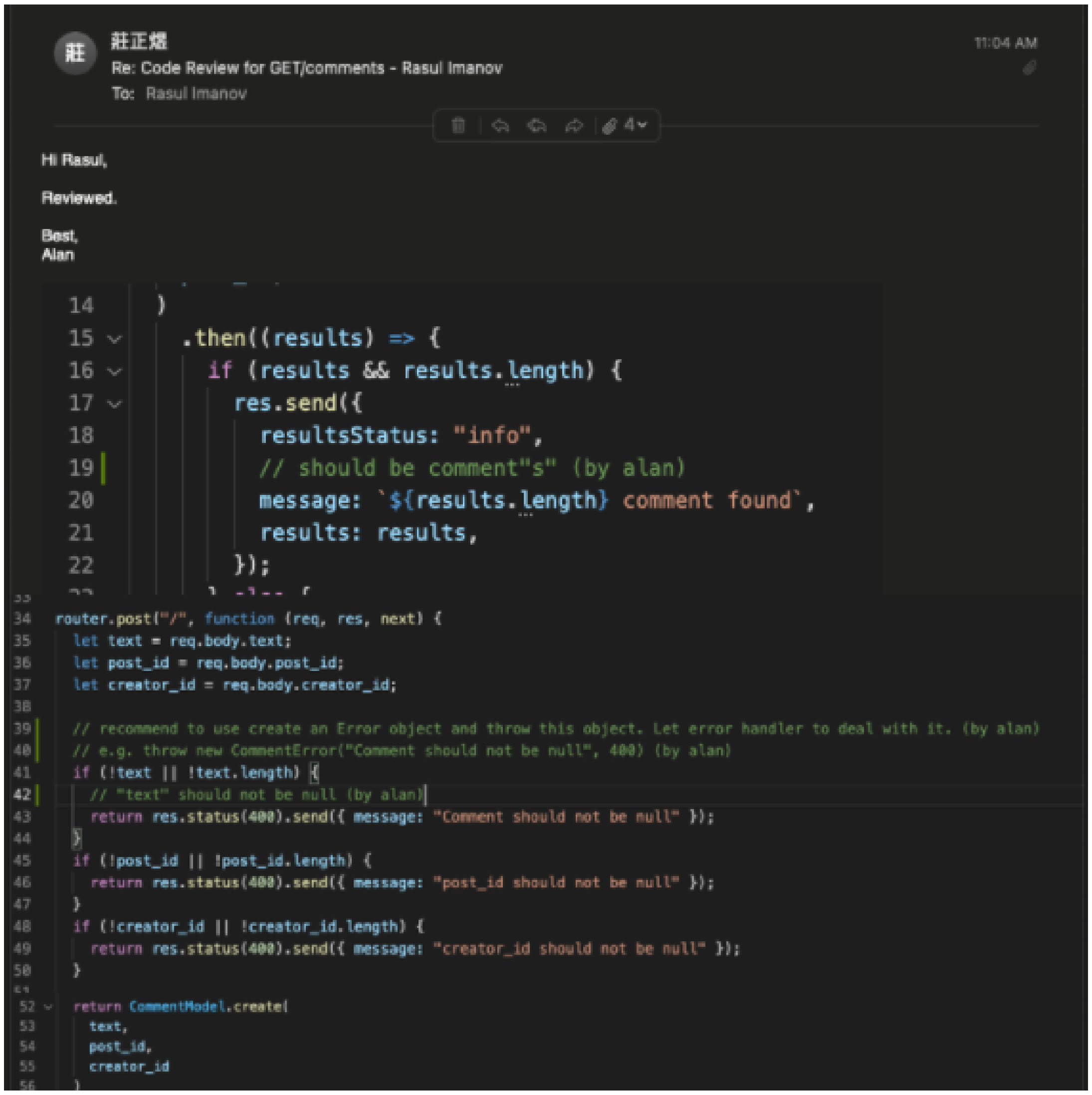
Hey Alan,

Would you please review my code and let me know if it looks good to you?

The code files I would like you to review are in
https://github.com/CSC-648-SFSU/csc648-fa21-02-team01/blob/master/application/rumi/backend/models/comments.js
https://github.com/CSC-648-SFSU/csc648-fa21-02-team01/blob/master/application/rumi/backend/routes/comments.js

I have also attached pictures of each file, so you are able to pick them apart

Thanks,
Rasul



莊 莊正煜
Re: Code Review for GET/comments - Rasul Imanov
To: Rasul Imanov

11:04 AM

Hi Rasul,

Reviewed.

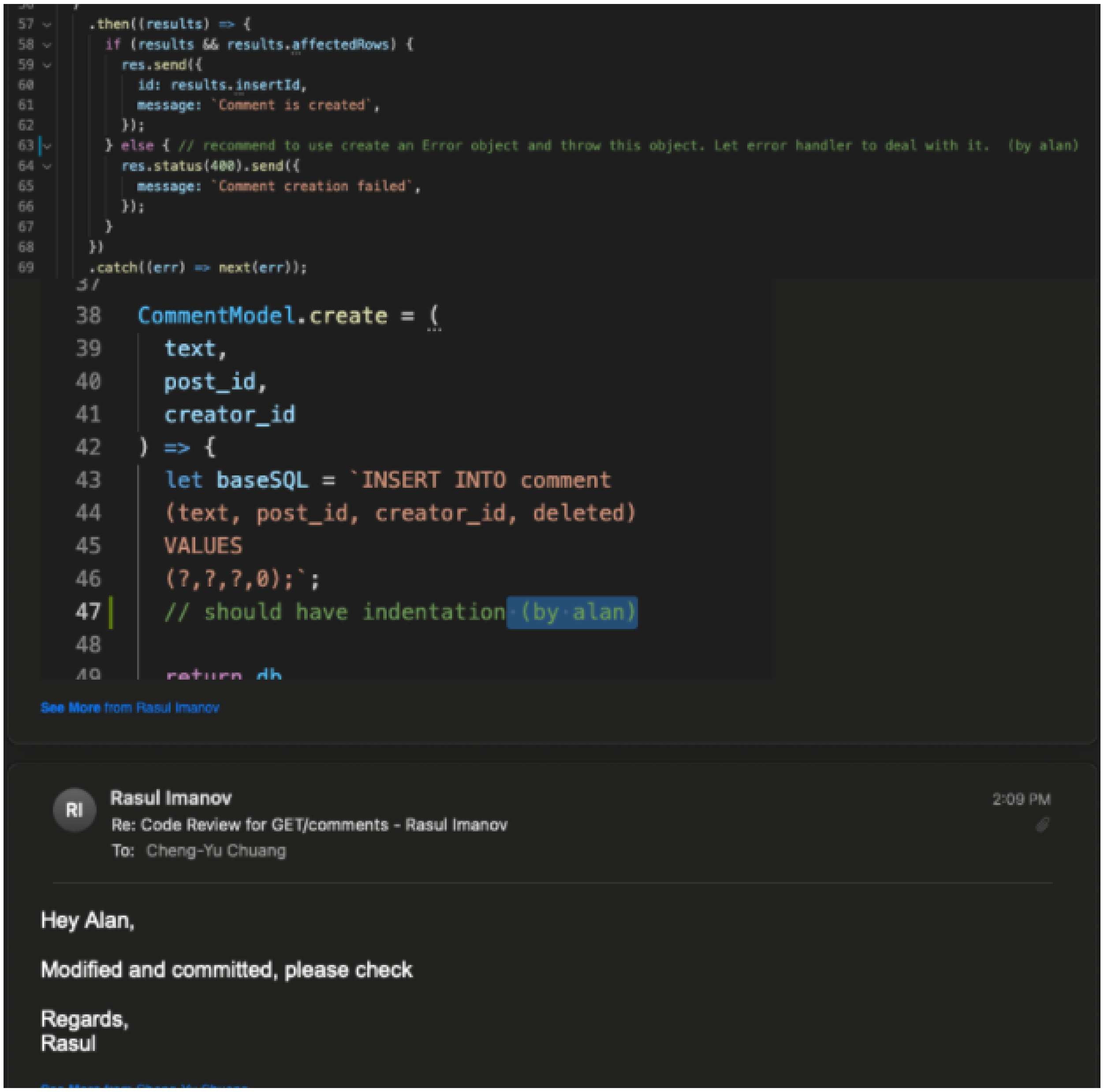
Best,
Alan

```
  )
    .then((results) => {
      if (results && results.length) {
        res.send({
          resultsStatus: "info",
          // should be comment"s" (by alan)
          message: `${results.length} comment found`,
          results: results,
        });
```

```
router.post("/", function (req, res, next) {
  let text = req.body.text;
  let post_id = req.body.post_id;
  let creator_id = req.body.creator_id;

  // recommend to use create an Error object and throw this object. Let error handler to deal with it. (by alan)
  // e.g. throw new CommentError("Comment should not be null", 400) (by alan)
  if (!text || !text.length) {
    // "text" should not be null (by alan)
    return res.status(400).send({ message: "Comment should not be null" });
  }
  if (!post_id || !post_id.length) {
    return res.status(400).send({ message: "post_id should not be null" });
  }
  if (!creator_id || !creator_id.length) {
    return res.status(400).send({ message: "creator_id should not be null" });
  }

  return CommentModel.create(
    text,
    post_id,
    creator_id
  )
```



```
  .then((results) => {
    if (results && results.affectedRows) {
      res.send({
        id: results.insertId,
        message: `Comment is created`,
      });
    } else { // recommend to use create an Error object and throw this object. Let error handler to deal with it.  (by alan)
      res.status(400).send({
        message: `Comment creation failed`,
      });
    }
  })
  .catch((err) => next(err));
```

```
CommentModel.create = (
  text,
  post_id,
  creator_id
) => {
  let baseSQL = `INSERT INTO comment
  (text, post_id, creator_id, deleted)
  VALUES
  (?,?,?,0);`;
  // should have indentation (by alan)

  return db
```

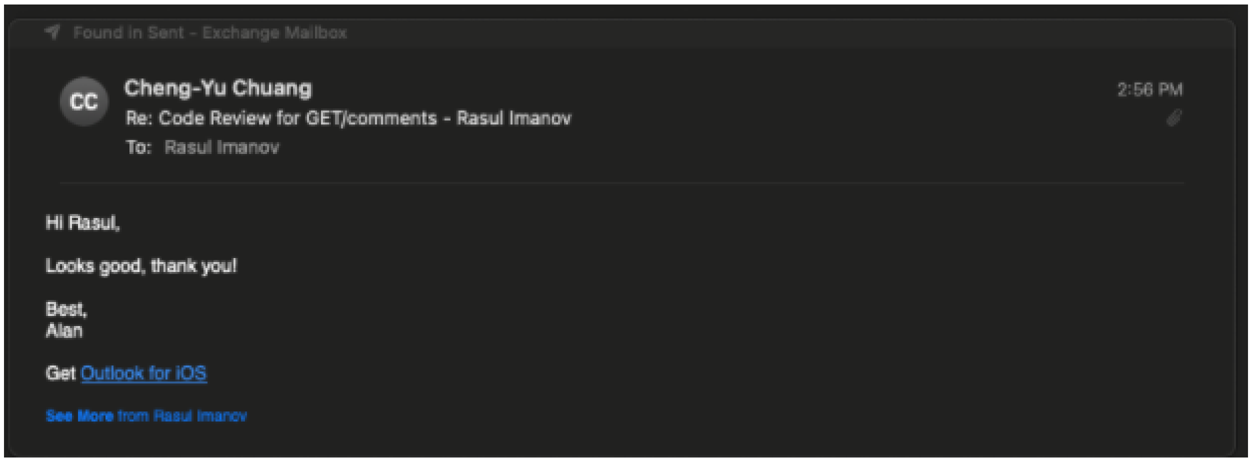
See More from Rasul Imanov

Rasul Imanov
Re: Code Review for GET/comments - Rasul Imanov
To: Cheng-Yu Chuang
2:09 PM

Hey Alan,

Modified and committed, please check

Regards,
Rasul



Found in Sent - Exchange Mailbox

Cheng-Yu Chuang
Re: Code Review for GET/comments - Rasul Imanov
To: Rasul Imanov
2:56 PM

Hi Rasul,

Looks good, thank you!

Best,
Alan

Get Outlook for iOS

See More from Rasul Imanov

5 Self-check On Best Practices For Security

➔ In this web application, HTTPS will be used instead of HTTP, since of course, HTTPS is more secure. While HTTP would be faster in terms of response times, it does not encrypt user data, meaning hackers that are positioned accordingly on the network can eavesdrop or view private customer data. Hence, HTTPS is valuable because it protects all communication and customer information. All the user's passwords are stored with encryption so that even in the case of a data breach, the intruders have access to encrypted passwords, which will be difficult to decrypt. Moreover, all our users will feel safer while browsing through our application.

➔ Our web application also ensures that all the data that goes from the user to our database is valid, hence comes the input data validation. Some of the data that is being validated are as:

➔ Front end

◆ **Register/Login -** Front-end is using npm packages like formik and yup, which validates all the inputs by the user in the registration and login forms.

For the registration form, front-end check for the conditions of -

- The username must not be empty, must be a string and must start with a character (a-z or A-Z) and have the length of at least 3 characters.
- Email must not be empty, must be a string and should be valid.
- The password must not be empty, must be a string and must pass the conditions of having the length be at least 8 characters long and must contain at least 1 upper case letter and 1 number and 1 of the following special character (/ * - + ! @ # $ ^ &).
- The input in the confirm password field should be exactly the same as the Password.

For the login form, front-end check for the conditions of -

- The username and password must not be empty and should be valid.

All these conditions must be passed by the user in order to submit these forms.

- **Protected Routes/Components -** We have also implemented protected routed and components on the front-end which protects certain routes/pages being accessed by a guest user. For example if a guest user tries to create a post, he/she will be directed to the login page. A user is only allowed to create a post when they are logged in. Another example of a protected component is that a guest user can't comment on any of the posts and is only allowed to see the comments.

➔ Back end

- **Create User -**
  Check not null and empty: username, email, password, description, Gender, School, major, smoker, pets
  Check email format:
  Regex: !/^\w{1,63}@[a-zA-Z0-9]{2,63}\.[a-zA-Z]{2,63}(\.[a-zA-Z]{2,63})?$/
  Check username format
  Regex:
  !/^[a-zA-Z].*/

- **Create Post -**
  Check not null and empty: caption, description, photoName, thumbnail, location, price, parking, pet, smoking, gender, creator_id, latitude, longitude

- **Create Comment -**
  Check not null and empty: text, post_id, creator_id

- **Create Message -**
  Check not null and empty: text, from_id, to_id

6 Self-check: Adherence to original Non-functional specs

➔ High-level non-functional specifications that MUST be adhered to

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)	**DONE**
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 concurrent users shall be accessing the application at any time	**DONE**
Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users	**DONE**
The language used shall be English	**DONE**
Application shall be very easy to use and intuitive	**DONE**
Google maps and analytics shall be added	**On Track**
No email clients shall be allowed. You shall use webmail	**On Track**
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	**On Track**
Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development	**DONE**
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. (Important so not to confuse this with a real application)	**DONE**