CSC 648/848 SFSU Fall 2023 Milestone 4

Title:

Team Number: Team 3

Beta Launch, QA and Usability Testing and Final Commitment for ProductFeatures (P1 list)

Date: 11/30/23

Team Member	Roles
Jeremy Tran	Team Lead / Github Master /jtran43@mail.sfsu.edu
Geovanni Valadez	Front End Lead gvaladez@mail.sfsu.edu
Mozhgan Ahsant	Scrum Master mahsant1@mail.sfsu.edu
Anthony Silva	Database Lead asilva32@mail.sfsu.edu
Aman Khera	Back End Lead akhera@mail.sfsu.edu
Ivan Ayala-Brito	Documentation Master iayalabrito@mail.sfsu.edu
Daniel Enriquez	Front End Engineer denriquez@mail.sfsu.edu
Alex Huang	Back End Engineer xhuang20@mail.sfsu.edu

History Table

Milestone Version	Date Submitted
M1V1	9/21/23
M1V2	10/12/23
M2V1	10/12/23
M2V2	11/2/23
M3V1	11/2/23
M3V2	11/30/23
M4V1	11/30/23

1. Product summary

Neon is a rental website for college students in San Francisco. Our product offers the standard features common to similar websites, including the ability for landlords to post and edit listings, and renters can search for listings based on their needs, such as cost, number of rooms, and location, to name a few. What makes our platform unique is the social aspect for our users and security checks for safety. Once a user has created an account, they can access the social page. From here, a user can create a profile, search for roommates, create groups, message users, and much more. Many times, students new to San Francisco need help to find a room to rent with people they trust and end up settling or finding themselves stuck to a lease in an uncomfortable living situation or, even worse, getting scammed. Our website addresses these issues with identity authentication and personality tests. With these tools, our users can find verified people with shared interests and develop trust before signing the lease, creating a safe and supportive home while completing their education.

2. Usability Test Plan

Map Region Selector

TEST:

- Function to send data from page to page,
    - What is being tested here is that each different region returns different values for our zip code
- function to retrieve data,
    - What is being tested here is to check if we are retrieving that data
- Function to determine if data retrieved or not, displays or not.
    - What is being tested here is, because there is more than one way to enter this page and select data (this is one specific way), it will allow you to still use the page with our without the data
- (Test description)
    - Start point, main page.
    - Intended users: all college students in the san francisco bay area

Search Area Code

TEST:

- Function to retrieve data based on given inputs
    - What is being tested here is that each input format might be different, so it checks for valid address first, then if not it'll show results based on zip entered
- RetrieveListing function
    - Parses inputs from user then lists actual data based on resulting conditions that are similar to input
- Error handling function
    - For frontend, this function will take place if the items in the search input are not valid, thus showing similar inputs to area code, or defaulting to all listings
- (Test description)
    - Start point, main page.
    - Intended users: all college students in the san francisco bay area
    - Measuring the effectiveness of the area codes returned (our district sections in database)

Set Status Hidden to Property

TEST:

- Function to select property
    - What is being tested here is that each listing can be selected, if multiple or just one can be selected at a time
- Function to make property hidden

        - What is being tested here is to see if we are making a property hidden from the listings database (not showing it).
    - (Test description)
        - Start point, social page.
        - Intended users: any home owners / realtors that wish to get rid of a specific listing
        - Measuring the effectiveness of hiding a property from the total properties
    - (Test description)
        - Start point, main page.
        - Intended users: all college students in the san francisco bay area
        - Measuring the listings displayed, whether it is any vs area code (area code is main way to see, but if user fails it doesn't lock or throw an error, allows them to continue and find other homes

Delete Property

TEST:

- Function to select property
    - What is being tested here is that each listing can be selected, if multiple or just one can be selected at a time
- Function to delete property
    - What is being tested here is to see if we are deleting the property from the user.
- (Test description)
    - Start point, social page.
    - Intended users: any home owners / realtors that wish to get rid of a specific listing
    - Measuring the effectiveness of deleting a property from their total properties

Social Page Lock

TEST:

- Function to retrieve login check
    - What is being tested here is that when attempting to click social, it checks to see if a user has a valid session id.
- Function to compare session id to user
    - This makes sure that the session id is actually from a user and not somehow fabricated
- Function to send to login or to social depending on result of proper session credentials
    - What is being tested here is that it will display visually differences to the user depending on whether or not they are logged in
- (Test description)
    - Start point, nav bar, so any page that isn't social page

- Intended users: all college students in the san francisco bay area and realtor homeowners that wish to put their properties on the site.
- Measuring the effectiveness of the lock, to ensure that not everyone will have access to this page.

Test	% completed	errors	comment
Map Region Selector	100%	None	Better for someone new to san francisco
Search Area Code	80%	Putting half an area code or a wrong area code doesn't give anything close, just default listings	If unfamiliar with other zip codes it won't give exact area
Set Status Hidden to Property	100%	Changes to inactive instead of hidden	Took too long to set hidden for some, but they all eventually got it
Delete Property	100%	None	Would've taken longer to set delete maybe if they hadn't done set hidden first (both similar process)
Social Page Lock	100%	None	Was intuitive for most, but one was confused why it took him to login page, which is what you need to do to access, which was understood for the rest

3. QA Test Plan

Test Objectives:

For our QA tests we want to ensure that our non functional requirements are able to get met and their are not data leaks or security bypasses are happening when our webapp is running. We are targeting specific tests to make sure their is storage capacity for all posts and listings and that we are able to correctly hash and salt user's password in the database. Additionally, we are going to allow users to provide feedback and ratings for their experiences on the web app such as their experiences with landlords or when seeing certain listings. Another test will be that we will test for there to be support for different database queries such as queries involving the search bar where listings with the similar addresses and the same zipcode can be found in the database. Finally, there will be tests for optimizing search functionality such as when using filters for searches and making sure that the database [rpvoded fast and accurate results, even with dealign with a large number of listings.

HW and SW Setup:

For all features, the hardware needed will be a computer or laptop with power, running on Windows, Ubuntu, or MacOS operating system which is connected to a stable internet connection. Users will need a browser such as Firefox, Chrome, or Safari installed on their computer. Last setup requirement is to naviage to a browser and go to our web app 54.198.188.175 which will have a domain name soon.

Features to be Tested

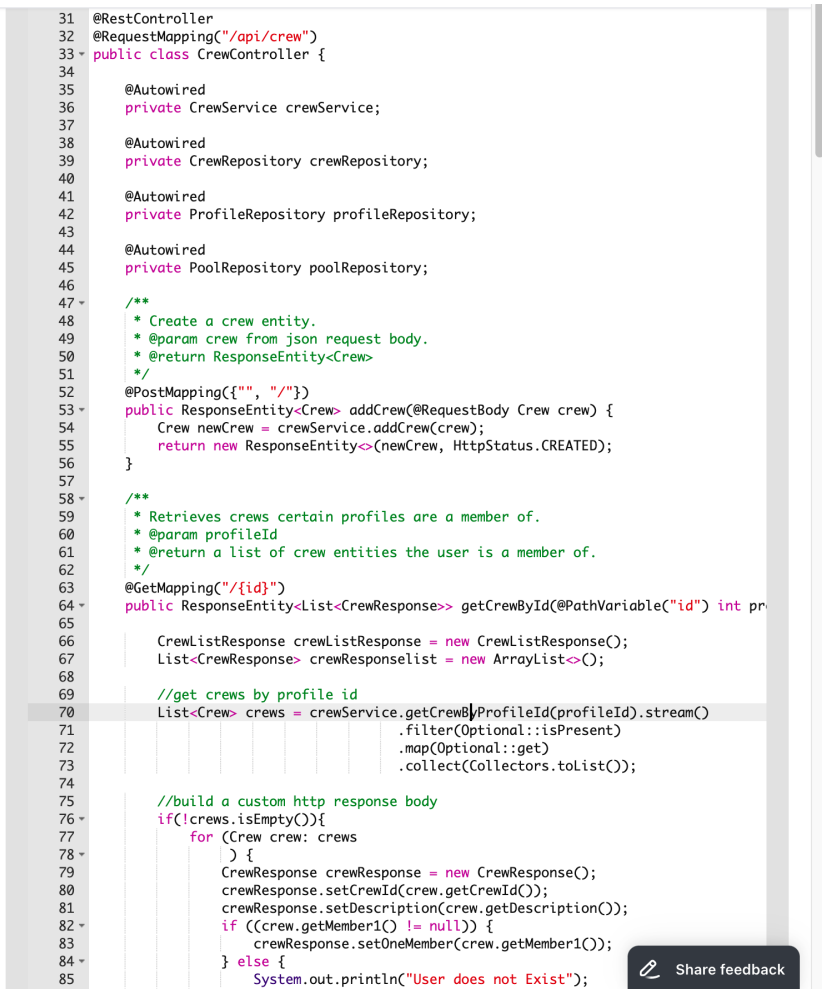
1. Storage capacity for all posts/listings
    1.1 Test to see the database storage for all the posts and listings that are added to the web
2. Hashing Password correctly after login
    2.1 Test to see that after a user enters a password, it is able to hash the password correctly in the database for that user's id
3. Feedback on Listings and Landlords
    3.1 Test to see that a user can provide feedback on rental listing and experience with landlord to help others make informed decisions.
4. Optimize search functionality even for large entries
    4.1 Tests to see that using filters for searches is efficient and making sure that the database provides fast and accurate results
5. Support of database index for fast access to table row data
    5.1 Test to see how different database queries such as queries involving the search bar where listings with the similar addresses and the same zipcode can be found in the database

QA Test Plan

Test #	Test Description	Test Input	Expected Correct Output	Test Results
1	Storage capacity for all posts/listings	Inserted post and listing	Success response and updated database	Pass
2	Hashing Password correctly after login	Login Details	Hased password in database	Pass
3	Feedback on Listings and Landlords	Feedback of listings or landlords	Stored data in database and confirmation of submission	Fail
4	Optimize search functionality even for large entries	Inputted filters for search query	Listings with the filteres applied	Pass
5	Support of database index for fast access to table row data	Inputted zipcode or address on searchbar	Listings with the exact zipcode or similar address	Pass

4. Code Review

Team 04 Code:



```
@RestController
@RequestMapping("/api/crew")
public class CrewController {

    @Autowired
    private CrewService crewService;

    @Autowired
    private CrewRepository crewRepository;

    @Autowired
    private ProfileRepository profileRepository;

    @Autowired
    private PoolRepository poolRepository;

    /**
     * Create a crew entity.
     * @param crew from json request body.
     * @return ResponseEntity<Crew>
     */
    @PostMapping({"", "/"})
    public ResponseEntity<Crew> addCrew(@RequestBody Crew crew) {
        Crew newCrew = crewService.addCrew(crew);
        return new ResponseEntity<>(newCrew, HttpStatus.CREATED);
    }

    /**
     * Retrieves crews certain profiles are a member of.
     * @param profileId
     * @return a list of crew entities the user is a member of.
     */
    @GetMapping("/{id}")
    public ResponseEntity<List<CrewResponse>> getCrewById(@PathVariable("id") int pr

        CrewListResponse crewListResponse = new CrewListResponse();
        List<CrewResponse> crewResponselist = new ArrayList<>();

        //get crews by profile id
        List<Crew> crews = crewService.getCrewByProfileId(profileId).stream()
                                      .filter(Optional::isPresent)
                                      .map(Optional::get)
                                      .collect(Collectors.toList());

        //build a custom http response body
        if(!crews.isEmpty()){
            for (Crew crew: crews
                 ) {
                CrewResponse crewResponse = new CrewResponse();
                crewResponse.setCrewId(crew.getCrewId());
                crewResponse.setDescription(crew.getDescription());
                if ((crew.getMember1() != null)) {
                    crewResponse.setOneMember(crew.getMember1());
                } else {
                    System.out.println("User does not Exist");
```

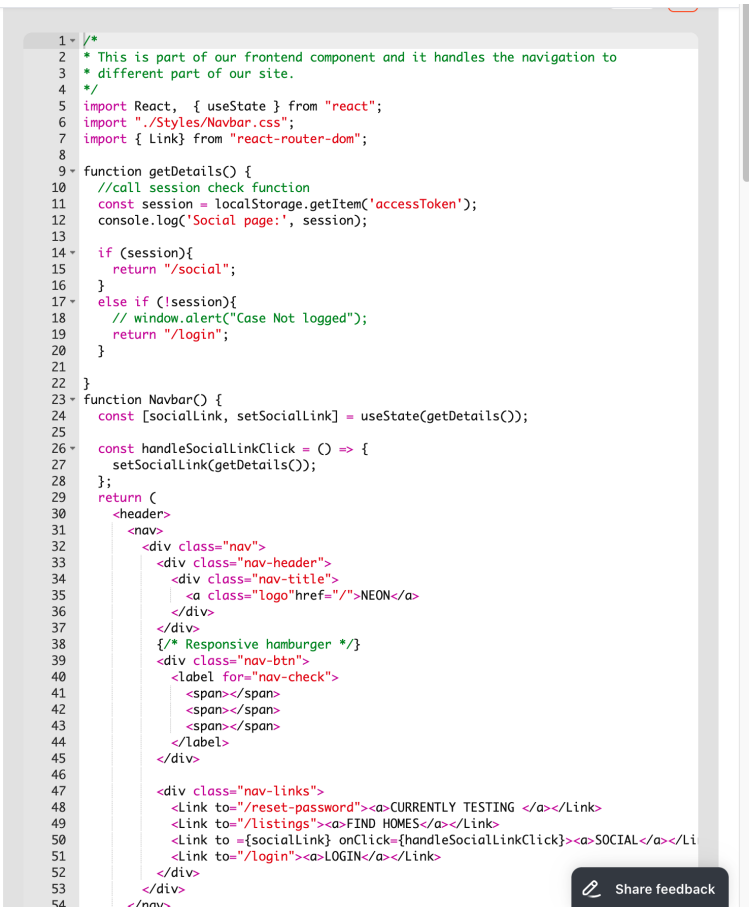
Share feedback

Code Review of 04:

“CrewController” class looks to be well-organized and does well at setting up endpoints for managing crews. I thought that all the methods are clear and well-commented. Overall, it was very easy to follow along and understand what was going on. There are a few areas that I think could be improved on.

- The error handling I think could be better such as on updateCrew where it assume the requested resource exists. It should handle cases where the resource isn’t there.
- Reponse from the server has a very general type ‘ResponseEntity<?>’. I think it could be more clear what kind of data type youre handling.
- Theres repeated code in createCrew, maybe make it into a function so it’s not duplicated.
- API paths are mostly consistent but ones like ‘/remove/member’ moves away from your pattern.
- There are use of ‘System.out.println’ which shouldn’t be there but I assume these will be removed during final submission.

Team 03 Code for 04: We went with the standard Java/JavaScript notation. We chose this snippet because it is one of the main component of our frontend and backend.



```
/*
* This is part of our frontend component and it handles the navigation to
* different part of our site.
*/
import React,  { useState } from "react";
import "./Styles/Navbar.css";
import { Link} from "react-router-dom";

function getDetails() {
  //call session check function
  const session = localStorage.getItem('accessToken');
  console.log('Social page:', session);

  if (session){
    return "/social";
  }
  else if (!session){
    // window.alert("Case Not logged");
    return "/login";
  }

}
function Navbar() {
  const [socialLink, setSocialLink] = useState(getDetails());

  const handleSocialLinkClick = () => {
    setSocialLink(getDetails());
  };
  return (
    <header>
      <nav>
        <div class="nav">
          <div class="nav-header">
            <div class="nav-title">
              <a class="logo"href="/">NEON</a>
            </div>
          </div>
          {/* Responsive hamburger */}
          <div class="nav-btn">
            <label for="nav-check">
              <span></span>
              <span></span>
              <span></span>
            </label>
          </div>

          <div class="nav-links">
            <Link to="/reset-password"><a>CURRENTLY TESTING </a></Link>
            <Link to="/listings"><a>FIND HOMES</a></Link>
            <Link to ={socialLink} onClick={handleSocialLinkClick}><a>SOCIAL</a></Li
            <Link to="/login"><a>LOGIN</a></Link>
          </div>
        </div>
      </nav>
```

Share feedback

Team 04 Code Review:
The Navbar component is well-structured and effectively handles navigation in your frontend. The organization of the code is clear, and comments provide helpful insights into the purpose of different sections. It's easy to follow along and understand the functionality.

- Responsive Design Considerations: While the responsive hamburger menu is a good addition, it would be beneficial to ensure that the associated CSS and media queries are well-optimized for various screen sizes. This ensures a seamless user experience across different devices.
- Accessibility Improvements: If the hamburger menu serves as a navigation feature, consider enhancing its accessibility (i.e. how can a screen reader identify the hamburger menu for a non-sighted end user?).
- Link Styling and Hover Effects: Introducing consistent styling and hover effects for navigation links enhances the visual appeal and user interactivity. This provides visual cues to users that these elements are interactive.

Team 05 Code:

```
def searchrestaurants(query, limit=None):
    try:
        # Using the 'ilike' function for a case-insensitive search
        restaurants = restaurant.Restaurant.query.filter(
            or(
                restaurant.Restaurant.name.ilike(f"%{query}%"),
                restaurant.Restaurant.address.ilike(f"%{query}%"),
                restaurant.Restaurant.cuisine.ilike(f"%{query}%")
            )
        ).all()

        restaurant_data = []
        for res in restaurants:
            res_img = restaurant_image.RestaurantImage.query.filter_by(restaurant=res

            img_url = None
            if res_img:
                img_url_obj = image_url.ImageURL.query.get(res_img.image)
                if img_url_obj:
                    img_url = img_url_obj.image_url

            restaurant_data.append({
                "name": res.name,
                "cuisine": res.cuisine,
                "address": res.address,
                "open_date": res.open_date,
                "rating": float(res.rating),
                "review_count": res.review_count,
                "image_url": img_url
            })

        return restaurant_data

    except Exception as e:
        print(f"Error fetching restaurants by query: {e}")
        return []
that whole function. it handles both the dropdown and the search page
we have 2 routes: 1 that handle the search_result page; and the other to handle the d
@bp.route('/api/search')
def search():
    query = request.args.get('search')
    restaurants = controllers.search_restaurants(query)
    return jsonify({"restaurants": restaurants})


@bp.route('/api/search_suggestions')
def search_suggestions():
    query = request.args.get('search')
    if not query:
        return jsonify({"error": "Invalid query parameter"}), 400

    try:
        restaurants = controllers.search_restaurants(query, limit=6)
        return jsonify({"restaurants": restaurants})
    except Exception as e:
```

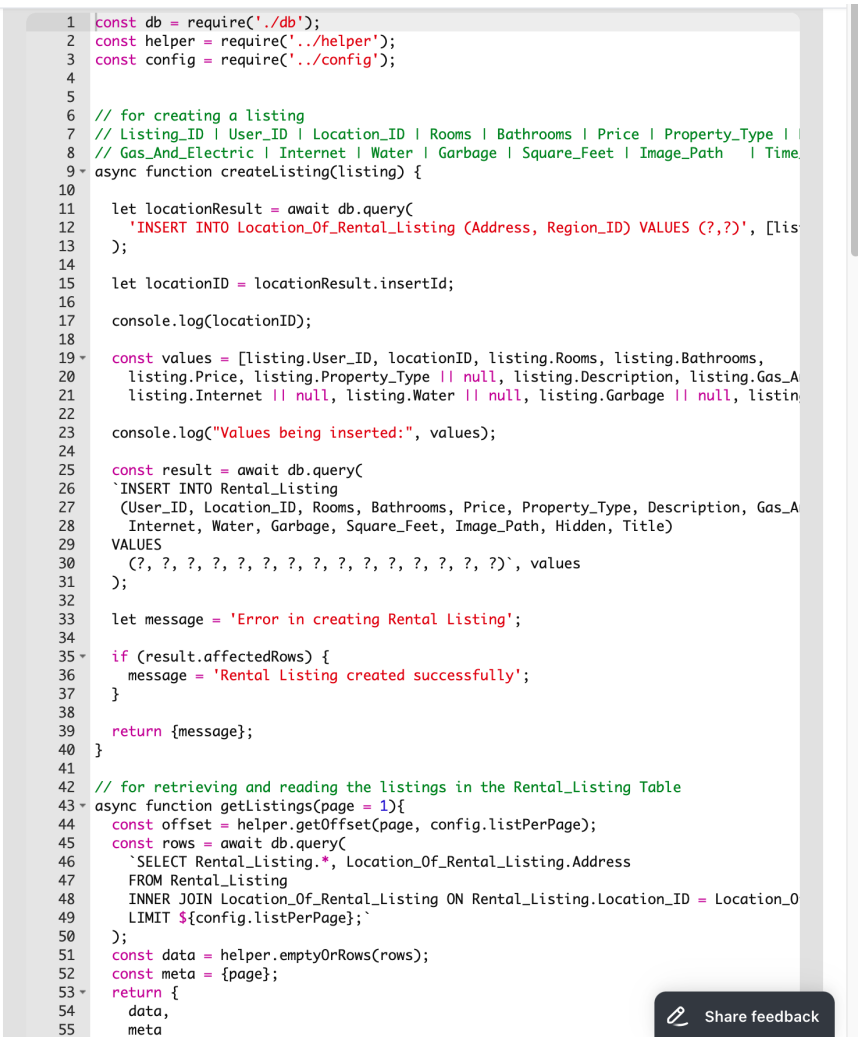
Share feedback

Code Review of 05:

Your search function seems to be efficient and uses the 'ilike' for case sensitive searches which makes it user friendly. Your process of iterating through each restaurant to gather relevant data is well-implmented in our opinion. An improvement you could make is the 'limit' parameter you mentioned isn't used in Searchrestaurant. This could be useful to control number of results.

The error handling is done well in your routes. Overall we think your implementation is done effectively. We feel as though we might've missed something or misunderstood so adding comments to your code could improve readability.

Team 03 Code for 05: Same for this snippet, we followed standard Java/Javascript notation. I choose this snippet because this involves a major component of our site which is rental listings and this handles all thing rental listings.



```
const db = require('./db');
const helper = require('../helper');
const config = require('../config');


// for creating a listing
// Listing_ID | User_ID | Location_ID | Rooms | Bathrooms | Price | Property_Type | |
// Gas_And_Electric | Internet | Water | Garbage | Square_Feet | Image_Path   | Time
async function createListing(listing) {

  let locationResult = await db.query(
    'INSERT INTO Location_Of_Rental_Listing (Address, Region_ID) VALUES (?,?)', [lis
  );

  let locationID = locationResult.insertId;

  console.log(locationID);

  const values = [listing.User_ID, locationID, listing.Rooms, listing.Bathrooms,
    listing.Price, listing.Property_Type || null, listing.Description, listing.Gas_A
    listing.Internet || null, listing.Water || null, listing.Garbage || null, listin

  console.log("Values being inserted:", values);

  const result = await db.query(
  `INSERT INTO Rental_Listing
   (User_ID, Location_ID, Rooms, Bathrooms, Price, Property_Type, Description, Gas_A
    Internet, Water, Garbage, Square_Feet, Image_Path, Hidden, Title)
  VALUES
    (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`, values
  );

  let message = 'Error in creating Rental Listing';

  if (result.affectedRows) {
    message = 'Rental Listing created successfully';
  }

  return {message};
}

// for retrieving and reading the listings in the Rental_Listing Table
async function getListings(page = 1){
  const offset = helper.getOffset(page, config.listPerPage);
  const rows = await db.query(
    `SELECT Rental_Listing.*, Location_Of_Rental_Listing.Address
    FROM Rental_Listing
    INNER JOIN Location_Of_Rental_Listing ON Rental_Listing.Location_ID = Location_O
    LIMIT ${config.listPerPage};`
  );
  const data = helper.emptyOrRows(rows);
  const meta = {page};
  return {
    data,
    meta
```

Share feedback

Team 05 Code Review:

Review for specific Functions: createListing:

- Efficiently handles the insertion of new listings into the database.
- Good use of template literals for SQL queries, though parameterized queries would be more secure against SQL injection.
- Consider handling exceptions with try-catch blocks for database operations.

getListings:

- Implements pagination, which is good for performance on large datasets.

- Directly interpolating `config.listPerPage` in the query might be risky. Ensure that this value is controlled and cannot be manipulated by end users.

updateListing:

- Uses a dynamic SQL query to update only provided fields. However, the use of template literals with direct variable insertion (`"${listing.User_ID}"`) is a serious security risk for SQL injection.

removeListing:

- Simple and effective function for deleting listings.
- Similar to `getListings`, ensure that `listing_id` is validated or comes from a trusted source to prevent SQL injection.

Concerns: The biggest concern is the potential for SQL injection, particular in your “updatingListing” function. Using template literals with variable interpolation in SQL queries (`\${variable}`) is not safe. try using parameterized queries or prepared statements to avoid this risk. Ensure that inputs (like listing_id, User_ID`, etc.) are properly validated before being used in database queries.

Summary: The code is well-organized and follows a logical structure for CRUD (Create, Read, Update, Delete) operations. Good use of async/await for handling asynchronous database operations. The code is generally readable, but adding more comments explaining the logic, especially in complex functions like “updateListing”, would be beneficial. Your code is functional with a clear approach to handling database operations. The primary area for improvement is security, specifically around preventing SQL injection.

5. Self-Check on best practices for security

```
mysql> SELECT * FROM Registered_User;
+-------------------+--------------------------------------------------------------+-------------------+--
| User_ID           | Password                                                     | email             | F
+-------------------+--------------------------------------------------------------+-------------------+--
| .useaspdm@uxf4ele | $2b$10$gxlIgHvL3kwBSfU8uZAlxOsC4OxQyJK0wP.yg11fnme/hm2.Q3Tzm | example4@sfsu.edu | T
| @lsue.mxfe2useapd | $2b$10$ZR.uqitDAuyriBiWqP5WgulqYKhH.z.ToS6UbgGDF6lXHetKe0WIy | example2@sfsu.edu | T
| alice             | 123456                                                       | user1@example.com | a
| bob               | qwerty                                                       | user2@example.com | b
| charlie           | password                                                     | user3@example.com | c
| d@eu.alesxpesuf3m | $2b$10$aQsBPLgos18rWuRht.sfhuqYm1oqdY0mgxGU2JteM2W61zEZHMJRm | example3@sfsu.edu | T
| ds@uausee.epx6mlf | $2b$10$lk1b4gcw3UgIYe1EAoDhVul1lrkd/Gn9u4qG1KVSl.d.Uubo9Pj/C | example6@sfsu.edu | T
| fu.edexeus1asmpl@ | $2b$10$1XlnSnXN3Q4zF8m6m.UGre42AI/RsyZIaZCmSvpRDT6qsr7CuSE0C | example1@sfsu.edu | T
| lmpciaexa1eemm@ol | $2b$10$vvtWRWkBmcYZjEorNmw8heUXQLDhiCL6vQXGME.lrDpQb6WTMOrMy | Jeff677@gmail.com | J
| mu@adlpxssufee5e. | $2b$10$AzKKwQrn6vbpbIFUScliw.l.F1WW/MQ4Eied8xWOTY7uS342VjzFy | example5@sfsu.edu | T
+-------------------+--------------------------------------------------------------+-------------------+--
10 rows in set (0.01 sec)
```

How we encrypt / decrypt the passwords:
We primarily use bcrypt to help with the encryption process. Retrieving password data for example when copmparing for a login, we await on bcrypt.compare() to see what is. When creating the user, we use hashing to be able to hash out every password and make sure it is encrypted, and whenever we have a password verification section such as login, we use decryptions bcrypt to check.

- Confirm Input data validation (list what is being validated and what code you used) – we request you validate search bar input;

For search bar input we use our function called retrieveListings. This is used for searching properties, and a zip code is generally required if the other data is not valid, so if a user tries putting invalid street or other malicious data, it won't apply and only returns the zip codes found or recent listings.

For what we validated data, for tier 2 verification, we validate all the data imputed to make sure it is proper data being inserted. It checks for proper email and phone number as well.

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

1. **Performance (DONE)**
    - The application shall provide high availability, minimizing downtime and ensuring accessibility to users.
    - The application shall provide fast response times for user interactions.
    - The system should support a minimum of 100 concurrent user sessions without a significant degradation in performance.
    - Optimize search functionality to deliver fast and accurate results, even when dealing with a large number of listings
    - The application shall have failover if an existing component fails or becomes unreachable.
    - The application shall support a database index for fast access to table row data and to improve query performance
2. **Storage (DONE)**
    - The application shall support storage capacity for all post/listing, and user interactions.
    - The application shall provide a mechanism for users to archive data.
    - Define how long different types of data (e.g., user profiles, listings, messages) should be retained and when it should be archived or deleted

3. **Security (DONE)**
    - The application shall enforce an access control mechanism.

- The application shall hash and salt the user's password in the database.
- User authentication should be robust, requiring secure credentials (e.g., passwords) and supporting multi-factor authentication (MFA) for added security.
- Maintain detailed logs of security-related events and user activities
- The application shall use firewalls to detect vulnerabilites and prevent data leakage

4. **Reliability (DONE)**
    - The website should be available 24/7, with planned downtime communicated to users in advance
    - The website should maintain reliability even during peak usage periods, ensuring consistent response times and availability
    - The system should have a documented and tested data recovery plan in place.
5. **Usability (DONE)**
    - Ensure the website is responsive and adapts to various screen sizes and devices, including mobile phones, tablets, and desktop computers.
    - Provide clear, concise, and well-organized content with easily readable text and appropriate use of headings, paragraphs, and lists.
    - Allow users to provide feedback and ratings for rental listings, helping others make informed decisions and fostering trust in the platform

- Allow users to report any misdemeanors or suspicious activty by landlords or renters

6. **Maintainability (DONE)**
    - Implement thorough testing procedures and automated testing suites to quickly identify and address issues during maintenance.
    - The website should be designed with a modular structure, allowing individual components to be updated or replaced without affecting the entire system.
    - Keep third-party libraries and dependencies up to date to avoid security vulnerabilities and compatibility issues.
7. **Data Privacy (DONE)**
    - All sensitive user data, such as personal information and payment details, must be encrypted both in transit and at rest to protect against unauthorized access.
    - Define and enforce data retention policies that specify how long user data will be stored and when it will be deleted or anonymized
    - Have a documented plan and process in place for responding to data breaches, including notifying affected users and relevant authorities as required by data protection regulations.
8. **Browser Compatibility (ON TRACK)**
    - The application should be compatible with a range of web browsers, including Chrome, Firefox, Safari.

- The website should adapt seamlessly to different screen sizes and devices, including desktop computers, laptops, tablets, and smartphones.
- Users should experience consistent functionality and visual design across different browsers, ensuring a uniform and user-friendly interface

9. **Preferred language (ISSUE)**

REASON: Reason for issue is that we do not have translations as the website is localized and we realized a prerequisite for students attending colleges in the san francisco bay area is to speak and understand english as all lectures. with the exclusion of classes about other languages, are taught in english.

- The website should support multiple languages, including but not limited to English, Spanish, French, and others based on user demand
- Users should be able to easily select their preferred language from a list of available options.

10. **Displaying name of website (ON TRACK)**

- Name of the website shall be shown on each page.

7. List of contributions

Members	Contributions	Score
Jeremy Tran	● Assisted and created multiple different APIs ● Fixed bugs in APIs ● Made changes to frontend to make pages be able to handle submits and handle data dynamically ● Did all the connection between APIs and UI components ● Was active in team calls to work with the group ● Did code review on the team's code	
Geovanni Valadez	● Organized the frontend team and the meetings ● Went through each page and started a list of tasks to fix ● Did section 2, 5 and 6 of M4 doc ● Fixed multiple bugs in the frontend ● Made changes to organize and add more color to UI ● Assisted the backend with some logic problem as well ● Turned in all his work and in a timely manner ● Was communicative and partook in discussions with the team ● Did M3 revisions ● Joined Team calls to work with the group	10
Mozhgan Ahsant	● Did the users, and location_of_rental_listing API ● Attended every meeting	4
Anthony Silva	● Attended every group meeting ● Made database changes to add new data variables we needed ● Add UI components: Tier 2 verification, background check status, housing cost calculator	5
Aman Khera	● Organized the backend team and set up timeline for APIs: searchBar, updateListing	9

	● Completed APIs: update profile page, phone verification ● Did the QA testing of the site ● Helped in the deployment of the application ● Turned in all his work and in a timely manner ● Was communicative and partook in discussions with the team ● Assisted in the connection between APIs and UI component ● Joined team call to work with the group	
Ivan Ayala-Brito	● Made small changes to orangize the code ● Attened all the team and frontend meetings	1
Daniel Enriqurez	● Created APIs: socialMedia, comments ● Attended all the team meetings	6
Alex Huang	● Made the Posts API ● Attended all the team meetings ● Was active in the team calls ● Made changes in the frontend based on Geo's task list	7