Final Project for SW Engineering Class CSC 648-848 Fall 2023

Team Number: 3

Neon

12/7/2023

http://54.198.188.175/

M5

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.

Priority 1:

1. **User:**

- (1.1) Users shall be able to register an account using a unique email and set up a password.

- (1.2) Users shall be able to browse the rental listings without an account but will not be able to interact with posts or other users.

- (1.3) The system should verify user identities securely through email or phone number verification.

2. **Background Check Integration**

- (2.1) The system shall integrate with predefined screening criteria and business rules, allowing property managers to set specific eligibility criteria based on factors such as credit score, criminal history, rental history, and income.

3. **Registered User**

- (3.1) Registered users shall be given the option to make posts on the site.

- (3.2) A Registered User shall be able to apply filters when searching for rental listings

- (3.3) A Registered User shall be able to filter specifically for number of bedrooms, bathrooms, and price.

- (3.4) A Registered User shall be able to use the search bar for retrieving listings based on a zipcode and address

4. **Location of Rental Listing**

- (4.1) Valid locations of rental listings shall show up on a map as markers so users can see where the location is at.

- (4.2) Rental listings shall be separated into regions, where users can view by clicking interactable map region.

5. **Rental Listing:**

- (5.1)Registered Users shall be able to update/edit their rental listing
- (5.2)Registered Users shall be able to delete their rental listing
- (5.3)Registered Users shall be able to specify number of bathrooms
- (5.4)Registered Users shall be able to specify number of bedrooms
- (5.5)Registered Users shall be able to specify number of price
- (5.6)Registered Users shall be able to specify address location
- (5.7)Registered Users shall be able to specify amenities in details
- (5.8)Registered Users shall be given the permission to add rental listing to the site.

6. **Housing Cost Calculator:**

- (6.1) The system will generate an approximation of yearly salary to takes to comfortably rent the rental listing.

7. **Posts:**

- (7.1) A user shall be able to create a post.
- (7.2) A user shall be able to attach a link in their post.

Features:

1. **Rental Listing Search by Region:** This function allows the user to search rental listing by clicking on a region which is the neighborhoods of San Francisco. So users can search listings by neighborhoods they want to rent in.

2. **Personality Test:** This feature involves a comprehensive set of questions designed to understand various attributes and preferences. It explores aspects like communication style, daily routines, hobbies, and preferences for cleanliness and organization. The system then uses the information to create compatibility scores between users. By focusing on these detailed personal traits, Neon ensures that potential roommates are well-matched in terms of their lifestyle and habits, significantly increasing the chances of a balanced shared living experience.

3. **Background Check:** This allows users to verify themselves a step further which will be displayed to other users. This is due to there being countless scams in the rental business, we want users to feel safe but not hindered if users choose not to verify. We use a 3rd party API called Authenticate to complete this process.

Milestone 1

1. Executive Summary

SFStudentRent is dedicated to creating a secure and supportive community for university students searching for housing compatible roommates near their campus. Safety is our top priority, recognizing that students often need a safe and secure environment as they embark on their journey towards higher education and independent living. Our platform offers a range of key features to enhance the student housing experience. Firstly, we provide an extensive selection of housing listings, making it easier for students to find suitable apartments and houses in close proximity to their campus. Whether they're seeking short-term rentals or more long-term arrangements.

SFStudentRent is more than just a housing marketplace; it's a social platform tailored for students. We encourage interaction and networking among students by enabling them to post and share housing opportunities. Moreover, user profiles on our platform can include links to social media profiles, allowing for even more extensive connectivity. We understand the importance of finding the right roommate, and our platform facilitates this by providing detailed bio pages. These pages offer insights into potential roommates personalities and preferences, ensuring students can find compatible living arrangements that suit their needs and lifestyle.

SFStudentRent aims to empower students, not only by simplifying their housing search but also by fostering a sense of community and security. Our platform is designed to help students make informed choices, build supportive living environments, and focus on their academic pursuits. Join us in creating a secure and enriching student housing experience.

2. Main Use Cases

1. **Use Case: Promote Real Estate Listing and Events**

Actor: Real Estate Agent/ Company

General User: Prior to the platform's launch, real estate agents mainly depended on conventional advertising channels like handing out flyers or using websites like Craigslist and Zillow to sell properties. On these sites, potential clients would often need to start direct contact with agents if they wanted information about open houses. However, this scenario has changed since the introduction of our platform. After signing up on our platform, an agent can now effectively market their properties on a site made for housing and interaction. They can list the property and open houses here along with other relevant housing events. Users can watch, RSVP to, or even forward questions regarding these events. The platform also offers a unique capability that enables agents to create focused groups. Agents can assemble their listings, updates, and events through these groups. Users can then join these groups whether they are interested in a specific agent's services or the overall scope of a real estate organization. By doing this, they are guaranteed to receive regular updates and have easy access to the agent's information. As a result, the platform not only makes it easier for agents to advertise, but it also centralizes information access for potential tenants or buyers.

2. **Use Case:** Tenant Verification for Landlords

Actor: Landlord

Landlord: Finding dependable tenants has traditionally been a risky and uncertain task for landlords. With our site, landlords are better equipped to handle this difficulty. Landlords can submit rental listings by including property information after creating an account on the platform. Once the listing is active, prospective tenants can

browse it, show their interest, and contact the landlord directly. As the inquiries come in, landlords don’t have to fear that these are fraudulent since all users on the platform are verified. This methodical investigation confirms the legitimacy of each prospective tenant's identity and background, providing the landlord with a condensed but thorough report without violating privacy laws. The platform also has a student verification process in recognition of the fact that some landlords have a particular preference for tenants who are students. As a result, the landlord has access to thorough, trustworthy information about each prospective renter, empowering them to make wise rental decisions. These functions let the platform act as a trusted intermediary between tenants and landlords, expediting the leasing process and encouraging real interactions.

3. **Use Case:** Registered User Verification
Actor: Registered User

Registered User: Before a user can fully engage with the site, beyond basic content viewing, they must complete a registration and verification process. This step ensures the user’s authenticity, a vital aspect when interacting with property listings from landlords. Once registered, users gain access to features like commenting, posting roommate requests, and sharing personal profiles to engage more extensively with the community and listing. This verification process not only fosters trust but also enhances the overall experiences, empowering users to make meaningful connections, find suitable accommodations, and showcase their personalities and preferences to the world.

4. **Use Case:** Registered User Friend Request
Actor: Registered User

Registered User: A registered user must complete the registration and verification process to unlock enhanced interaction features on the site. Once registered and verified, users gain the ability to send friend requests to other registered users. This feature promotes a sense of security, as the site’s background checks ensure that all

users are genuine individuals. Users can confidently connect with others who share similar qualities and goals, through a suggested friends tab that will suggest users who are similar to what you might like ,such as both being interested in door activities and searching for apartments in the same area. This shared compatibility makes adding friends an appealing and productive step in the process of finding potential roommates. Users can also search for a user individually to send a friend request, they can click their post or profile picture to get to their profile tab to then send them a friend request. By sending a friend request, registered users initiate a connection that allows them to explore the possibilities of living together, fostering a community where like-minded individuals can pursue their housing goals with confidence.

5. **Use Case:** Landlord, New listing

Actor: Landlord (John)

Description: John is a landlord looking for someone to rent his 1 bedroom house. He had already registered on our website. So, he logs into his account, and clicks on the new listings button. He will be redirected to a new page and enter his home information in detail. His home data could be his address, some image, price, number of bedrooms and bathrooms, date available, his email or his phone number. Finally, after making the new listings, he enters the submit button to publish his new listings. Therefore, his new listing will be displayed on our website.

6. **Use Case:** Unregistered User,

Actor: Melissa

Description: Melissa, an unregistered user seeks to rent a house or apartment. Upon visiting our website, she encounters a user-friendly interface featuring a search bar and powerful filtering options. While pursuing,she notices advertisements for apartment rentals. Melissa’s target is an apartment in California’s Bay Area. She initiates her search by entering the zip code of her desired location into the search bar. The website promptly generates a list of apartments and houses, each with varying

price points. To refine her search further, Melissa engages the filter feature. She specifies her preferences, opting for a one-bedroom, one-bathroom apartment, and sets price limits. With affordability in mind, she arranges the results from low to high prices. Upon discovering an interesting listing, Melissa endeavors to contact the landlord. However, the website reminds her to register as a user. Complying with this requirement, she provides her first name, last name, email address, and password, effectively becoming a registered user, enabling her to connect with the landlord effortlessly.

7. **Use Case:** Tenant, new listing

Actor: Registered User (Lisa)

Description: Lisa had a challenging year with her previous roommate, enduring mental stress, financial difficulties, and landlord disputes. Despite being the primary leaseholder, she struggled to enforce lease obligations and house rules. When the year-long lease ended, her landlord allowed her to stay and search for a roommate. Lisa, burdened by her demanding work schedule, turned to our website for help. Within just 30 minutes, she created an account and posted a listing. Shortly after, she found a compatible roommate. During the period between posting the listing and finding a roommate, our website sent notification to members who matched her criteria. This streamlined the search process made communication between Lisa and applicants effortless. Today, Lisa and her new roommate are content with their living arrangement, appreciating the ease and effectiveness of our services.

8. **Use Case:** Unregistered user, apartment seeking

Actor: Unregistered user (Tom)

Description: Tom, a busy professional preparing for a relocation to a new city many states away, faced the daunting task of finding a comfortable and secure place to

live in an unfamiliar culture. In his search for a rental room, he scoured numerous renting websites, only to be left dissatisfied. While these platforms provided extensive listings, complete with photos, amenities, and pricing, they sorely lacked critical information about potential roommates. Determined to make an informed choice, Tom persevered in his search until he stumbled upon our website. Here, his confidence in making a decision was significantly bolstered. As a newcomer to the area, he grappled with legitimate concerns about the possibility of being taken advantage of. However, after creating an account on our platform, Tom scrolled through the media feed to see what was being promoted. Tom found another working professional also trying to save on money posting looking for roomates on the main feed. Tom opened the other users profile to find that they had similar matching qualities other than both being working professionals, with their budgets being similar and desired location as well. Tom was able to see that this person was a real person as well through their social media pages being linked to the profile and being a verified through there. Tom found himself matched with a trustworthy roommate. Today, he enjoys a sense of security and contentment in his new city, all thanks to the positive experience facilitated by our platform.

9. **Use Case**: Searching the area

Actor: Registered user (Billy)

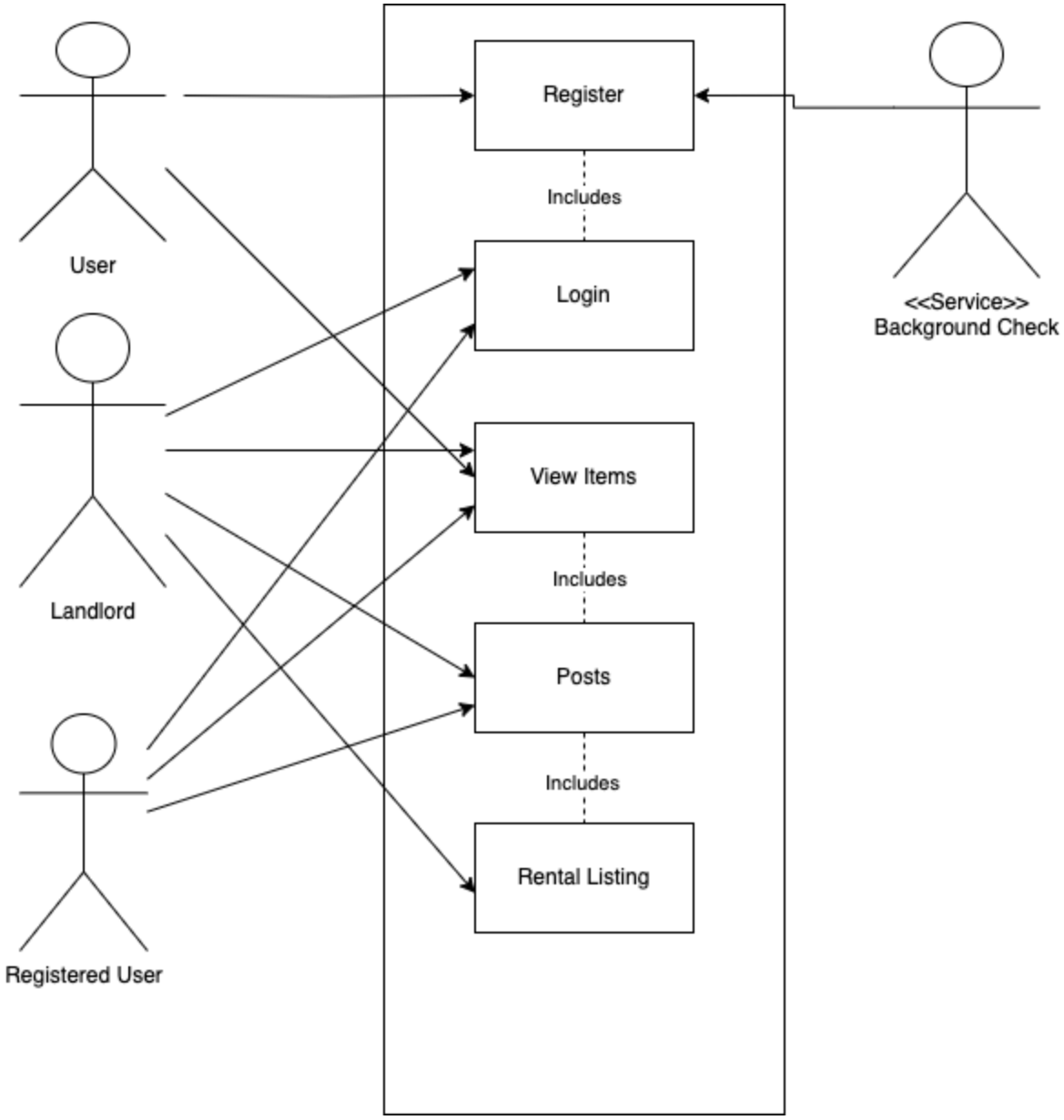
Description: Meet Billy, a regular user in our website to find a place to rent, he starts by typing in the area where he wants to rent, and our website helps him by suggesting places as he types. To make his search more specific, Billy can use filters, specifying how much he's willing to pay, how many bedrooms he wants and what extras he'd like in the place. When Billy sees a map on our site with markers, he can click on them to see the places available in those spots. It's like clicking on a map to see houses for rent in different neighborhoods. Here's the cool part. Billy can also draw a custom area on the map, and our side will show him places available in that area. It's saying "I

want to see rentals only in this part of town”. Billy is getting closer to finding his ideal rental place with every click.

10. **Use Case**: Setting up open houses

Actor: John (landlord)

Description: John, a landlord, is utilizing our website to connect with potential renters. His enthusiasm to showcase his property led him to create an open house listing on our platform. This comprehensive post includes crucial information such as the property’s location, its appearance, and the scheduled viewing times. Prospective visitors can easily express their interest by signing up for the open house directly through the website. This sign-up process enables John to stay informed about who intends to visit his property. As the open house data approaches, both John and registered attendees receive timely notifications. These messages serve as reminders and confirmations, ensuring that everyone is well-prepared and eager for the upcoming event. This streamlined process enhances the landlord’s ability to showcase his property effectively and provides a smooth experience for potential renters.


Register
Includes
Login
View Items
Includes
Posts
Includes
Rental Listing
User
Landlord
Registered User
<<Service>>
Background Check

3. List of Main Data Items and Entities w/ Attributes

1.User:

- UID (User ID)

2. Background Check:

- CheckID

3. Registered User:

- UID (User ID)
- Email
- Name
- Username
- Filters

4. Renter Account:

- RenterID

5. Location of Rental Listings:

- Maps
- Filter
- Search

6. Rental Listings:

- ListingID
- Bedrooms
- Bathrooms
- Price
- Location
- Filters
- Time and Date
- Home Type
- Option to add more

7. Posts:

- Post ID
- Time and Date

8. Messages:

- MessageID
- Message time and date
- CommentID
- Time and Date

9. Notification:

- NotificationID

10. Events:

- EventID
- Time and Date

11. Friend Request:

- Date and time of the request
- Status
- RequestID

12. Saved Properties List:

- Number of houses
- Similarities in houses
- Pricing
- ListID

13. Housing Cost Calculator:

- Number of residents
- Duration of lease
- Expected growth of the area
- Interest

14. Groups:

- GroupID
- Name

15. Reports:

- ReportID
- Time and Date

4. Initial list of functional requirements

1. **User**
    - Users shall be able to register an account using a unique email and set up a password.
    - Users shall be able to browse the site without an account but will not be able to interact with posts or other users.
    - A user can reset their password if forgotten.
    - The system should verify user identities securely through email or phone number verification

2. **Background Check Integration**
    - All registered users that wish to rent or rent out their place shall be subject to a background check.
    - Registered users can open the status of the background check.
    - The system shall integrate with predefined screening criteria and business rules, allowing property managers to set specific eligibility criteria based on factors such as credit score, criminal history, rental history, and income
3. **Registered User**
    - Registered users shall be given the option to make posts on the site.
    - Registered users shall save rental listings to their saved properties list.
    - Registered users shall send and receive friend requests.
    - A register user shall be able to create one free renter Account 8
    - A Registered User shall be able to form groups of other registered users they wish to house with 7
    - A Registered User shall be able to link their social media accounts such as Instagram and Twitter

4. **Location of Rental Listing**

- Locations shall include campus area, neighborhood, and proximity to public transport.
- Rental listings shall be separated into regions, where users can view by clicking interactable map region
- Users should be able to filter and view listings by clicking on map markers or drawing custom search areas.
- Users may opt to receive notifications when new rental listings by distance from a specified location(e.g., within 5 miles of a particular address or point of interest)
- The system shall provide autocomplete suggestions to assist users in specifying the location

5. **Rental Listing**
    - Registered users shall be able to create, update or delete their rental listings, specifying details such as number of bedrooms, bathrooms, etc.
    - Each rental listing shall have a unique ListingID.

6. **Posts**
    - Posts shall be only edited by the author.
    - A post shall be able to contain text, multimedia content(photos, links) and other metadata.
    - Posts shall allow comments, each with a unique commentID and creation timestamp.
    - A post shall be able to get pinned by the creator with a maximum of three for his or her account
    - A post shall be able to get promoted by the creator at least twice with the boost lasting two days

7. **Messages**

- Registered users shall be able to initiate private messages with one other registered user.
- The system should allow users to organize their messages, including archiving or deleting old messages
- Users should receive real-time notifications(e.g., push notifications or email notifications) when they receive a new message or when there are updates to existing messages

8. **Notification**
    - Each notification shall have a unique NotificationID.
    - Registered users shall receive a notification upon receiving a new message.
    - A notification shall be able to be turned off or on by a register user or group through notification settings in the future
9. **Events**
    - Registered users shall be able to set up events(open house, etc) for other users to see.
    - An event shall have its own location, date, eventID, and a list of confirmed attendees.

10. **Friend Request**
    - Registered users can send friend requests to connect with other registered users.
    - Every friend request shall have an unique requestID, date of creation, date and time of approval or denial, and status.
    - Register users sending the friend request shall allow permission changes for the requesting friend to view content.
11. **Saved Properties List**
    - Users can create lists of saved properties.
    - Each saved list shall have a unique ListID.
    - Properties List shall be under a registered user and not viewable to other non registered or registered users 9

12. **Housing Cost Calculator**
    - Registered Users can generate approximate costs for chosen intervals and forecasts for the property 5
    - The Calculator shall generate a curated list of houses best fit for a registered user based on filters and location 6
13. **Groups**
    - A group shall share a saved properties list and all receive notifications for a property 11
14. **Reports**
    - Registered users can generate reports on rental listings, user activity, and feedback.
    - The system shall provide functionality to generate various types of rental reports, including rental transaction summaries, rental history for specific customers, and rental equipment utilization reports
    - Users shall be able to customize the content and format of reports, including selecting specific date ranges, sorting options, and the inclusion/exclusion of specific rental data field, such as rental duration, customer information, and equipment details

5. List of non-functional requirements

1. **Performance**
    - 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**
    - 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**
    - 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**
    - 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**
    - 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**
    - 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**
    - 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**
    - 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**
    - 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**

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

6. Competitive analysis

Feature/Company	Zillow	Realtor.com	Apartments.com	Craigslist	Facebook Marketplace
Strengths	Many listings, User-Friendly Interface. Zillow has an intuitive design that makes it easy for users to navigate.	Good hook on landing page, follows good design patterns,, high quality photographs, offers many services related to real estate	Has nice map layout with many listing, gives so much information about each property with virtual tours;Clean UI; a lot of filters for properties	Many listings	Many listing, smooth UI/UX, lots of filter option, posts come from accounts
Weakness	Reliance on User-Generated Data. User-generated content can lead to inaccuracies and outdated listings.	Business generally oriented to buying and selling property,listing on search engines is very low when looking for apartments	Incostenient promoting of properties, less functionality for landlords, cannot regulate interactions between landlords and homeowners, anyone even without an account can harass or call landlords	Poor design, quality check on listings	Listings are missing information, some users have weak accounts as in very new and no profile
Pricing	Free for consumers	Free for renters and paid services for realtor	Free, with paid background checks	Free	Free
Social Media	Twitter, Instagram,	Facebook, Linked In,	Instagram, Twitter, TikTok	None	Twitter, Instagram

	TikTok, Facebook	Instagram, Twitter, Pinterest, You Tube			
Onboarding Experience	User-Friendly Sign-Up, Property Alerts. zillow allowing users to sign up using their email address or social media accounts	Instantaneous access to information about home and rental properties without requiring signing up or logging in. Automatic listings based on users location	Good for renters; asks for location and give properties with a map and prices with many filters. Landlords have to find where to add a property and not as user-friendly or intuitive for them. Also does not prompt making an account or signing in when first accessing the site.	Overwhelming, very simple, reliability	Simple, sign up/log in page is the first things you see

Feature	Zillow	Realtor.com	Apartment.com	Craigslist	Facebook	**Our future product**
Text Search	**++**	**++**	**+**	**+**	**+**	**+**
Boolean Search	**+**	**-**	**-**	**+**	**-**	**+**
Browse	**++**	**++**	**++**	**+**	**+**	**+**

Shopping Cart	+	-	-	-	-	-
Social Media Integration	-	-	-	-	+	++

Our future product will have the capability of searching for properites such as its competitors in an efficient and location-based manner with many filters and details in every search. What will differentiate our product is how it will allow for users to find compatible roomates who are background checked and with build in social media links can easily connect with future roomates. Every other company either has a property or roommate finder but none of them have yet to integrate both in one platform. For the property finder aspect we will priortize user experience by nofifying anytime a detail of a property changes, if a open house is available, and recommended a curated list of properties that best fit the needs of a user. A lot of these user-based needs are missing in our competitors and they rely solely on filter-based searching. We will also take into account the needs of landlords and provide securities from them which are not present in competitors. For example, making sure landlords have features as well to enhance their search of users, and adding a security layer where only a registered and interester buyer can contact them. Also, in the platform will incorporate a way to report misdeanmors between buyers or landlords and any suspicious activities which is missing in many of our competitors. With the social media component, integration of property and roomate search, user-friendly design and principles, and securities for landlords as well as renters our product will provide the most complete and satisfying experience on the market.

7. System Architecture and Technologies

1. **Server and Hosting**
   - Cloud Provider: Amazon AWS
   - CLI: AWS CLI V2

2. **Operating System:**
   - OS: Ubuntu 22.04.03 LTS

3. **Database:**
   - RDBMS: MySQL 8.0

4. **Web Server:**
   - Server: Nginx 1.18

5. **Development:**
   - Server-Side Language: JavaScript
   - Web Framework: React
   - IDE: Visual Studios Code

6. **Security:**
   - SSL: Amazon Certificate Manager
      - Not yet available due to missing domain
      - Unable to find free service for domain name

8. Checklist

• Team found a time slot to meet outside of the class: **DONE**
• Github master chosen: **DONE**
• Team decided and agreed together on using the listed SW tools and deployment server: **DONE**
• Team ready and able to use the chosen back and front end frameworks and those who need to learn are working on learning and practicing: **DONE**
• Team lead ensured that all team members read the final M1 and agree/ understand it before submission: **DONE**
• Github organized as discussed in class (e.g. master branch, development branch, folder for milestone documents etc.): **DONE**

9. List of team contributions M1

Names	Contributions	Rating
Jeremy Tran	● Set up instance and hosted server ● Created the skeleton for the About page ● Finished functional requirement, use case, competitive analysis and within timely manner ● Organized Documentation ● Filled out Section 7 of document ● Created Discord server dedicated for communication before access to #TEAM03 and now for chat that doesn't involve the whole team	
Geovanni Valadez	● Organized and set up document ● Finished functional requirement, use case, competitive analysis and executive summary ● Finished a working About page ● Very responsive and involved with the group ● Attended every group meeting ● Collected time availability of each group member ● Finished all work on time	10
Mozhgan Ahsant	● Finished functional requirement, use case, competitive analysis within timely manner ● Very communicative and asked for help when needed ● Responsive and involved with the group ● Finished About page ● Checked over document	8

	• Has attended every group meeting	
Anthony Silva	• Finished use case, competitive analysis • Finished About page • Assisted with database set up in instance • Helped check server side set-up • Responsive and involved with the group • Finished all task within timely manner and communicated if couldn't	9
Aman Khera	• Finished functional requirement, use case and competitive analysis • Finished About page within timely manner • Is researching about SSL • Has attended every group meeting • Been responsive and involved with the group • Checked over the document and left comments where needed work done	8
Ivan Ayala-Brito	• Did functional requirements and use cases • Checked over the document • Reorganized and reformatted documents • Finished About page • Communicative through private chat	6

Milestone 2

1. Data Definitions

1.User

- Unregistered: Limited to view-only access.
- Key Attitributes: N/A

2. Background Check

- Assesses registered user's financial and rental history.
- Key Attributes: Check_ID, Status

3. Registered User

- Full site access with functionalities like posting, messaging and joining groups.
- Key Attributes: User_ID, Password, Check_ID

4. Location of Rental Listings

- Precise location for rental listings.
- Key Attributes: Location_ID, Listing_ID

5. Region

- General location of the rental listing such as neighborhood.
- Key Attributes:Region_ID, Location_ID

6. Rental Listings

- A post that contains media, images or videos along with information regarding the property.
- Key Attributes: Listing_ID, User_ID

7. Posts

- Generic posts with or without media.
- Key Attributes: Post_ID, User_ID, Comment_ID

8. Messages

- Private communication between registered users.
- Key Attributes: Message_ID, User_ID(to), User_ID(from)

9. Notification

- Updates for users about thei account activities.
- Key attributes: Notification_ID, Notification_Type

10. Events

- User-created events either open or by invitation.
- Key attributes: Event_ID, User_ID(Host)

11. Friend Request

- Registered users can add other registered users to their friend list.
- Key Attributes: Request_ID, User_ID(to), User_ID(from)

12. Saved Properties List

- Registered users can save preferred rental listings.
- Key Attributes: List_ID, Saved_ID

13. Housing Cost Calculator

- Provides rental/buying suggestions based on input.
- Key Attributes: N/A, info not saved

14. Groups

- For group communications and activities.
- Key Attributes: Group_ID, User_ID

15. Reports

- Allows for users to report rental listings.
- Key Attributes: Report_Id, User_ID

2. Prioritized Functional Requirements

Priority 1:

1. **User:**

- (1.1) Users shall be able to register an account using a unique email and set up a password.

- (1.2) Users shall be able to browse the site without an account but will not be able to interact with posts or other users.

- (1.3) A user can reset their password if forgotten.

- (1.4) The system should verify user identities securely through email or phone number verification.

2. **Background Check Integration**

- (2.1) All registered users that wish to rent or rent out their place shall be subject to a background check.

- (2.3) The system shall integrate with predefined screening criteria and business rules, allowing property managers to set specific eligibility criteria based on factors such as credit score, criminal history, rental history, and income.

- (2.2) Registered users can open the status of the background check.

3. **Registered User**

- (3.1) Registered users shall be given the option to make posts on the site.

- (3.2) A Registered User shall be able to link their social media accounts such as Instagram and Twitter.

4. **Location of Rental Listing**

- (5.1) Locations shall include campus area, neighborhood, and proximity to public transport.

- (5.2) Users should be able to filter and view listings by clicking on map markers or drawing custom search areas.

- (5.5) Rental listings shall be separated into regions, where users can view by clicking interactable map region.

5. **Rental Listing:**

- (6.1) Registered Users shall be able to create, update, or delete their rental listings, specifying details such as the number of bedrooms, bathrooms, etc.

- (6.2) Registered Users shall be given the permission to add rental listing to the site.

6. **Housing Cost Calculator:**

- (13.1) Registered Users can generate approximate costs for chosen intervals and forecasts for the property.

7. **Posts:**

- (7.2) A post shall be able to contain text, multimedia content (photos, links), and other metadata.

- (7.3) Posts shall allow comments, each with a unique commentID and creation timestamp.

- (7.1) Posts shall be only edited by the author.

Priority 2:

1. **Registered User:**

- (3.2) Registered users shall save rental listings to their saved properties list.

- (3.7) A Registered User shall be able to form groups of other registered users they wish to house with.

2. **Rental Listing:**

- (6.3) Rental Listing shall have a minimum amount of information and media needed for posting.

3. **Messages:**

- (8.1) Registered users shall be able to initiate private messages with one other registered user.

4. **Saved Properties List:**

- (12.1) Users can create lists of saved properties.

- (12.2) Each saved list shall have a unique ListID.

- (12.3) Properties List shall be under a registered user and not viewable to other non-registered or registered users.

- (12.4) Properties list shall group properties by similarities or by a key word as desired by a registered user.

5. **Groups:**

- (14.2) A group shall share a saved properties list and all receive notifications for a property.

6. **Reports:**

- (15.1) Registered users can generate reports on rental listings, user activity, and feedback.

7. **Housing Cost Calculator:**

- (13.2) The Calculator shall generate a curated list of houses best fit for a registered user based on filters and location.

8. **Reports:**

- (15.2) The system shall provide functionality to generate various types of rental reports, including rental transaction summaries, rental history for specific customers, and rental equipment utilization reports.

- (15.3) Users shall be able to customize the content and format of reports, including selecting specific date ranges, sorting options, and the inclusion/exclusion of specific rental data field, such as rental duration, customer information, and equipment details.

Priority 3:

1. **Registered User:**

- (3.3) Registered users shall send and receive friend requests.

2. **Location of Rental Listing:**

- (5.3) Users may opt to receive notifications when new rental listings by distance from a specified location (e.g., within 5 miles of a particular address or point of interest).

- (5.4) The system shall provide autocomplete suggestions to assist users in specifying the location.

3. **Posts:**

- (7.5) A post shall be able to get pinned by the creator with a maximum of three for his or her account.

- (7.6) A post shall be able to get promoted by the creator at least twice with the boost lasting two days.

4. **Messages:**

- (8.4) The system should allow users to organize their messages, including archiving or deleting old messages.

- (8.5) Users should receive real-time notifications (e.g., push notifications or email notifications) when they receive a new message or when there are updates to existing messages.

5. **Notification:**

- (9.2) Registered users shall receive a notification upon receiving a new message.

- (9.3) A notification shall be able to be turned off or on by a register user or group.

6. **Events:**

- (10.1) Registered users shall be able to set up events (open house, etc) for other users to see.

- (10.2) An event shall have its own location, date, eventID, and a list of confirmed attendees.

- (10.3) An event shall only be made by verified registered users and have an expected list of attendees.

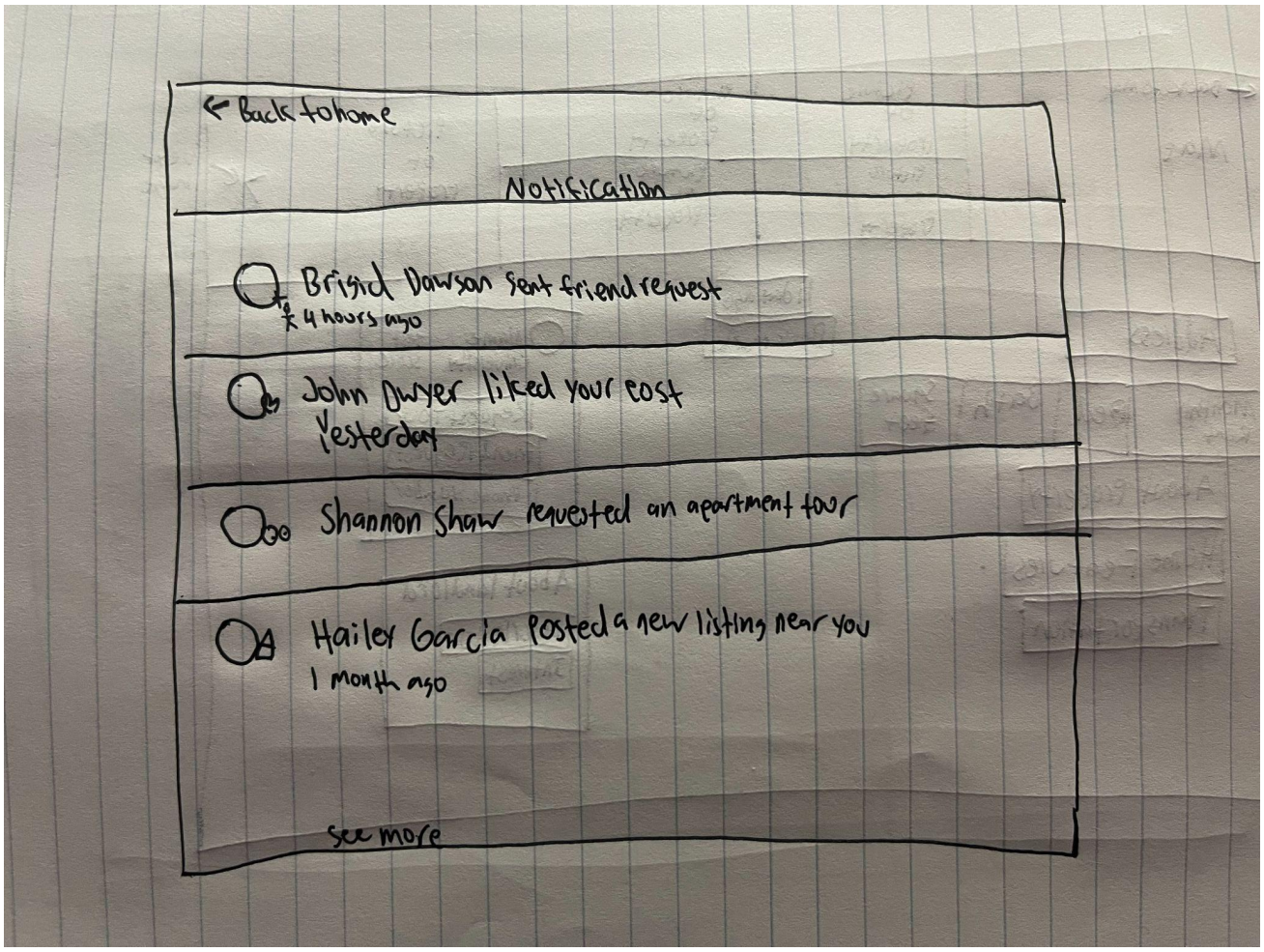
7. **Friend Request:**

- (11.3) Register users sending the friend request shall allow permission changes for the requesting friend to view content

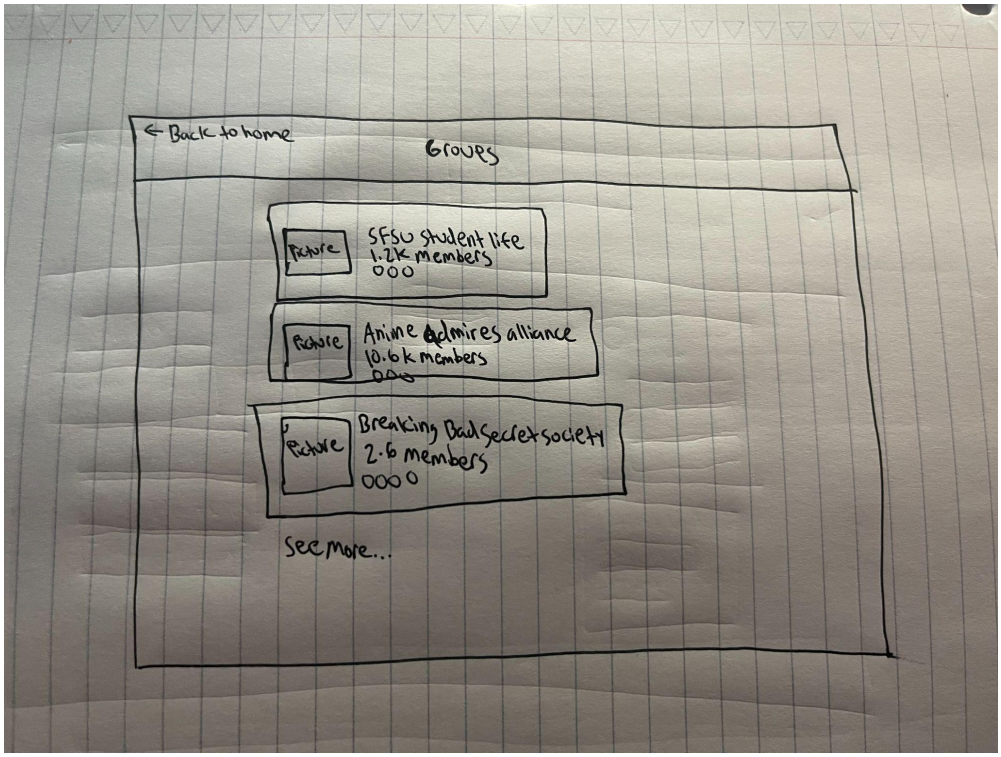
3. UI Mockups and Storyboards (high level only)

MOCKUPS:

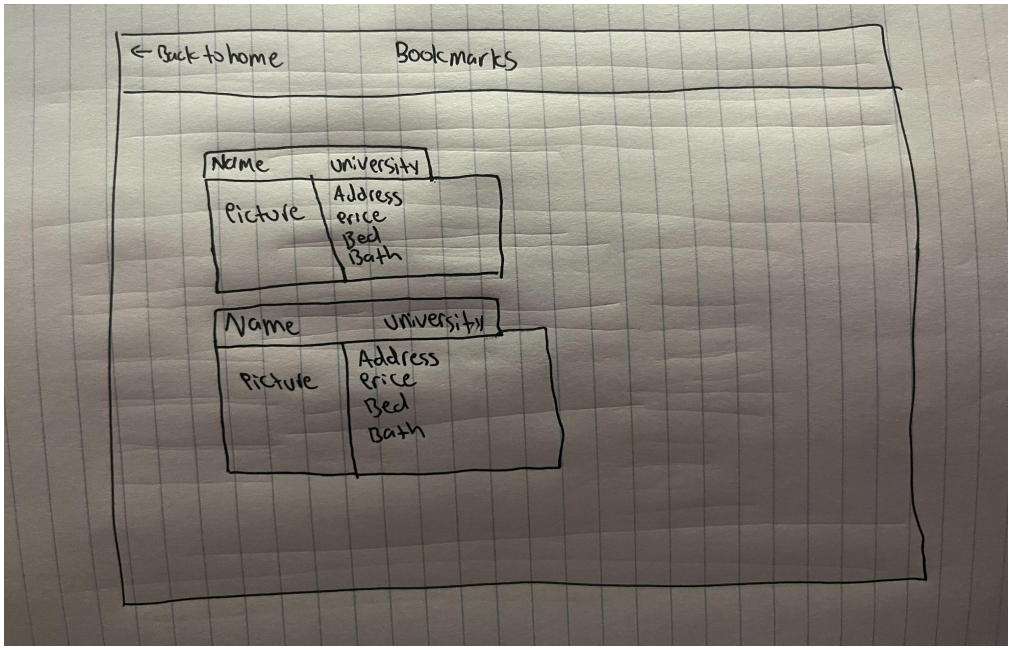
Notification List:



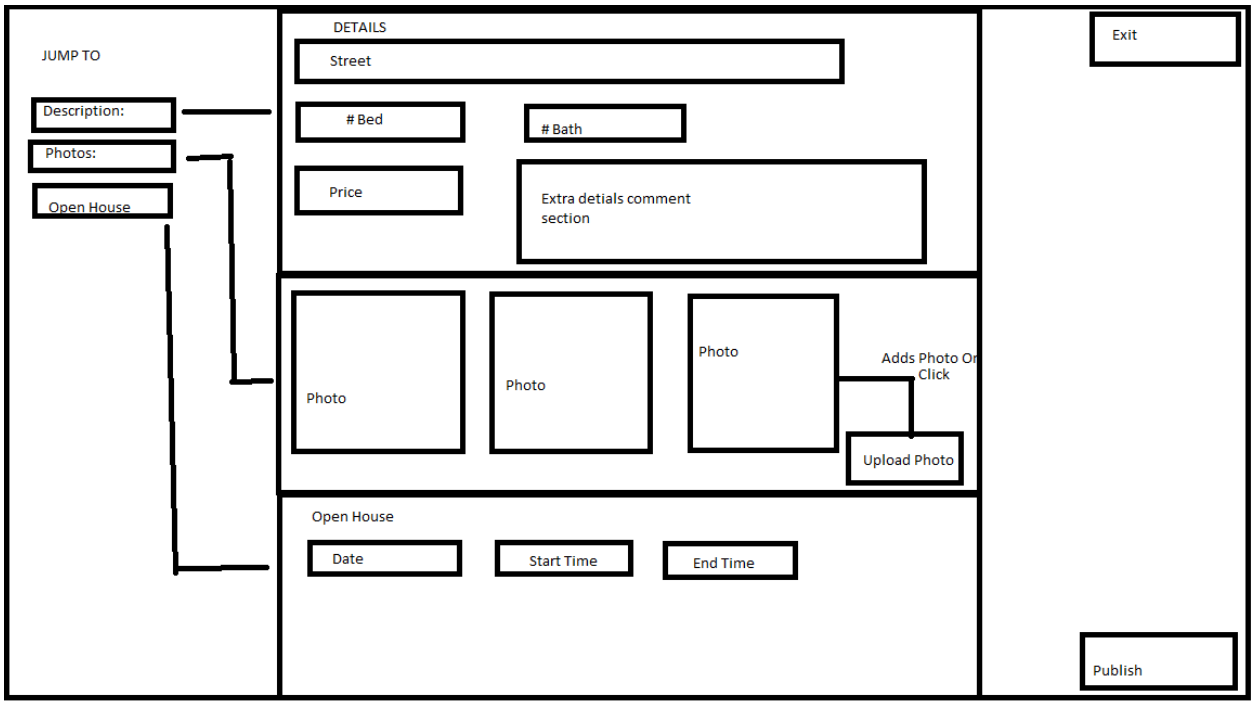
Groups:



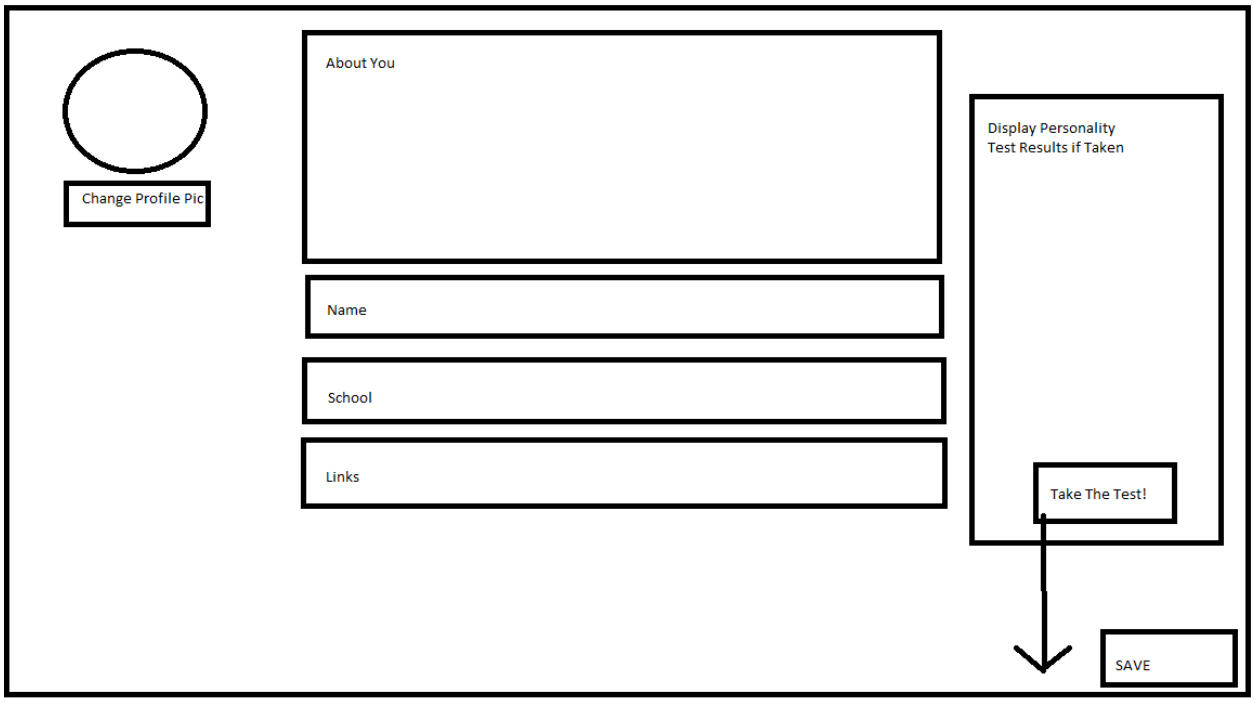
Bookmark:



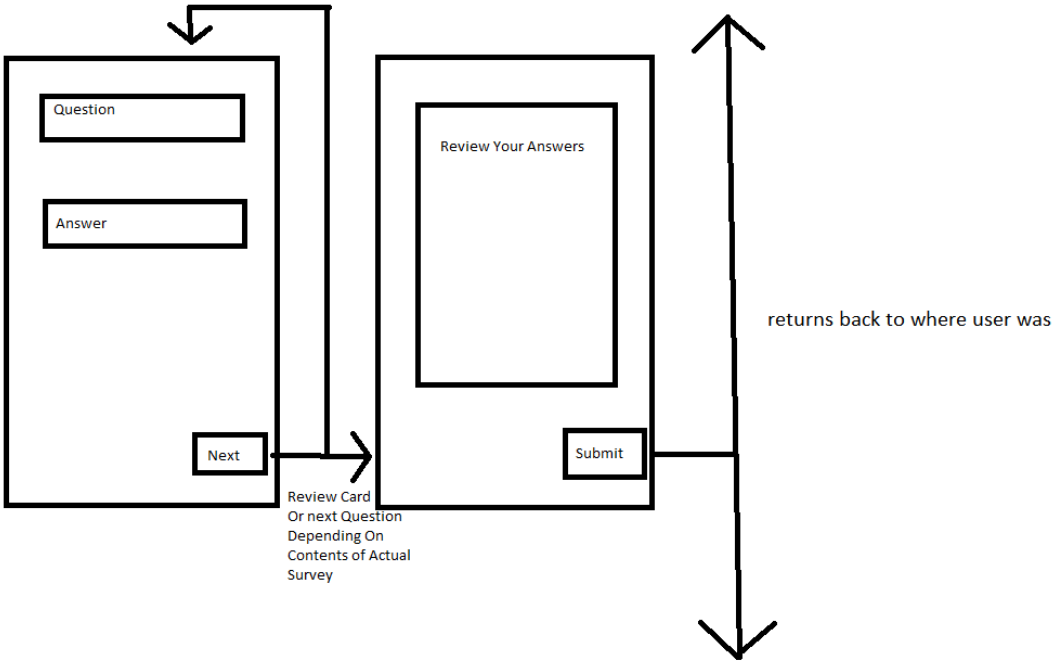
Edit Listing:



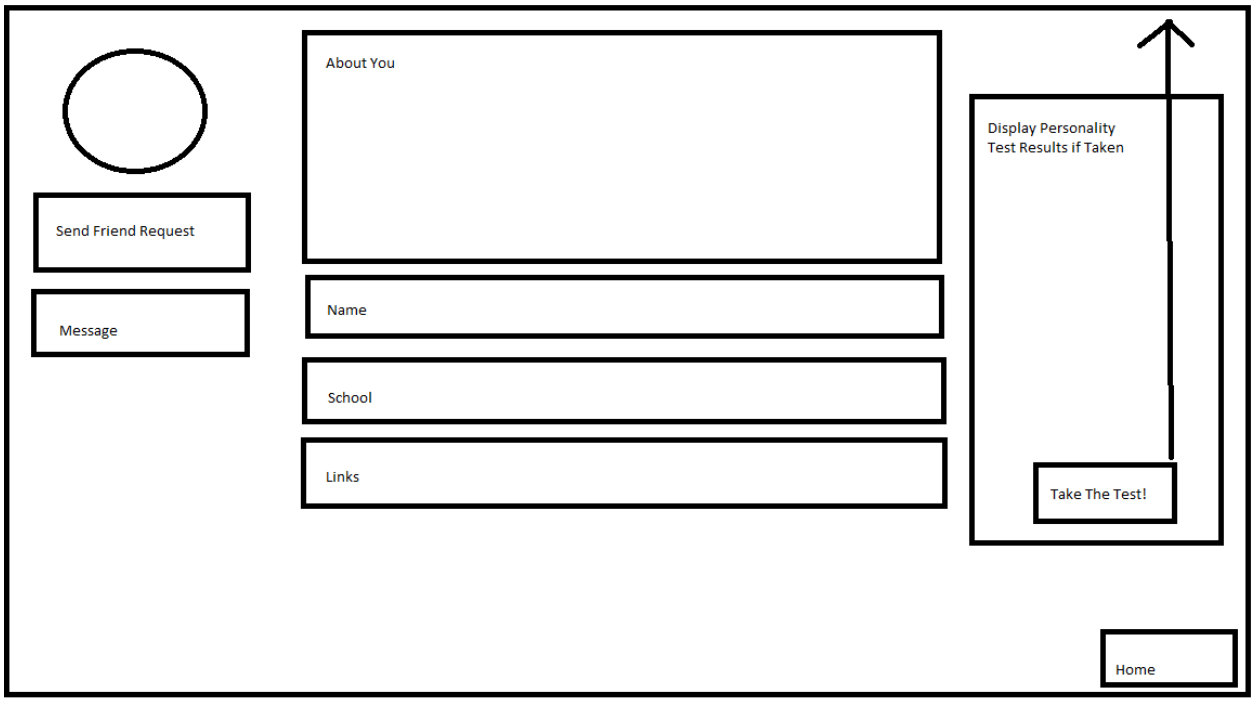
Edit Profile:



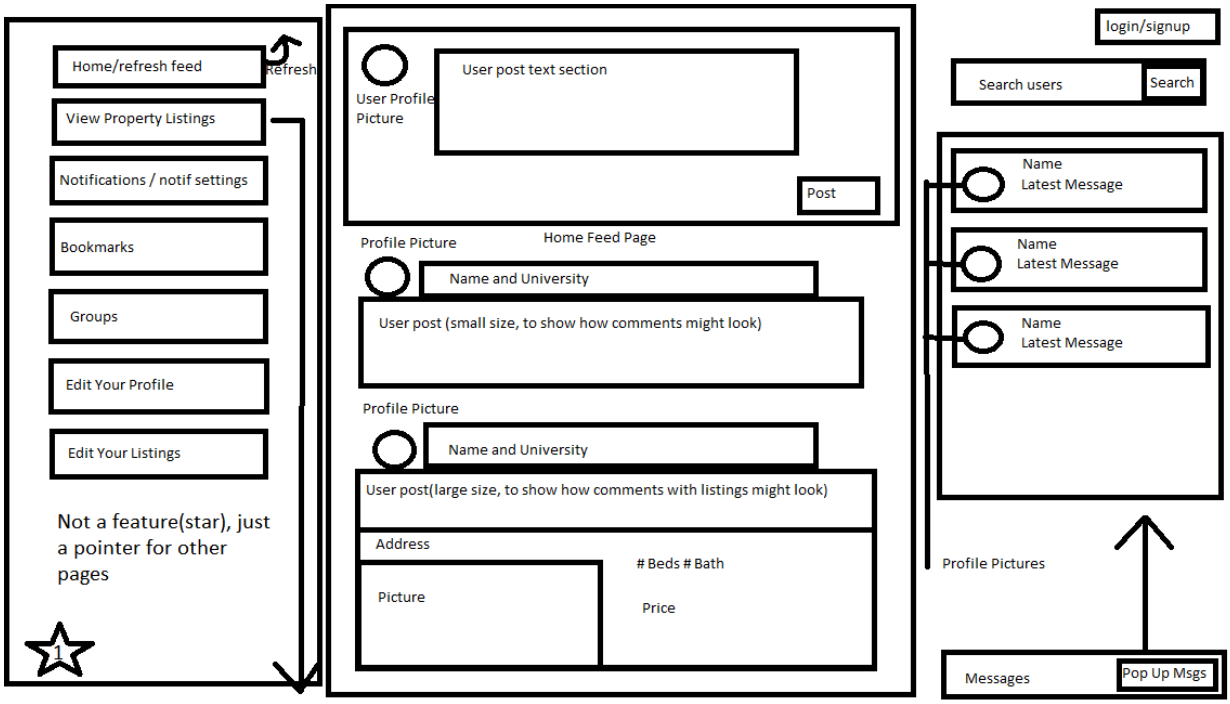
Personality Test:



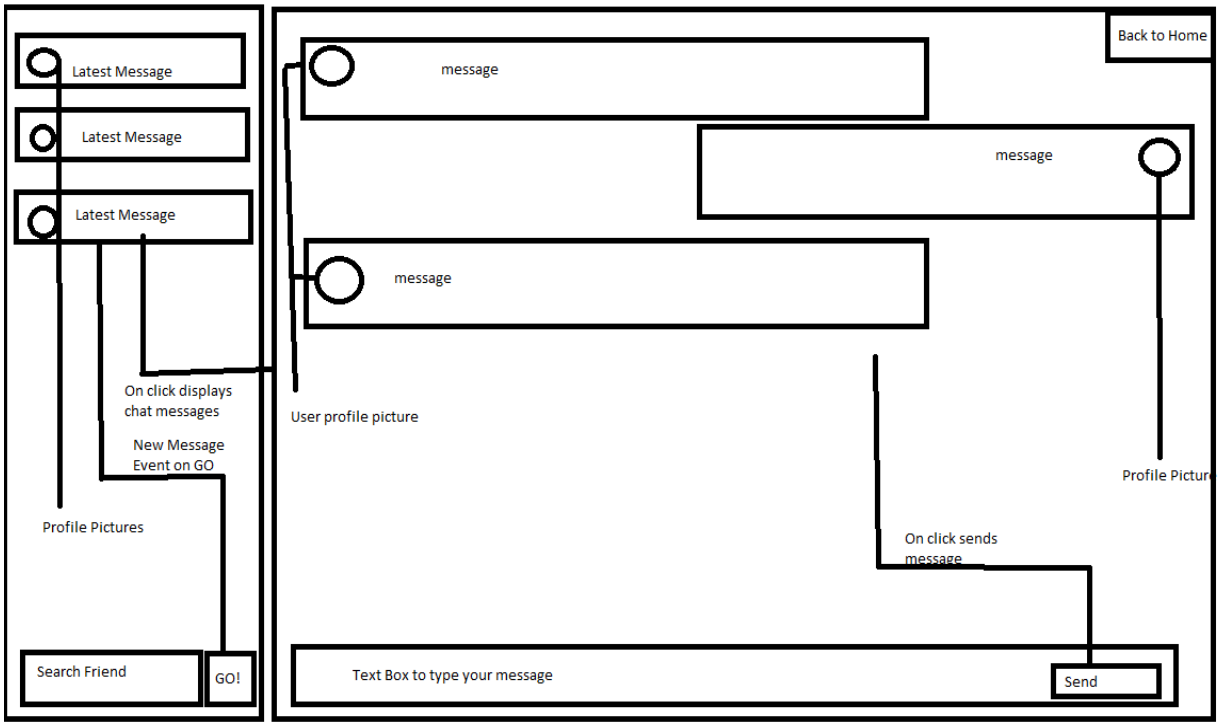
View Profile:


About You
Send Friend Request
Message
Name
School
Links
Display Personality
Test Results if Taken
Take The Test!
Home

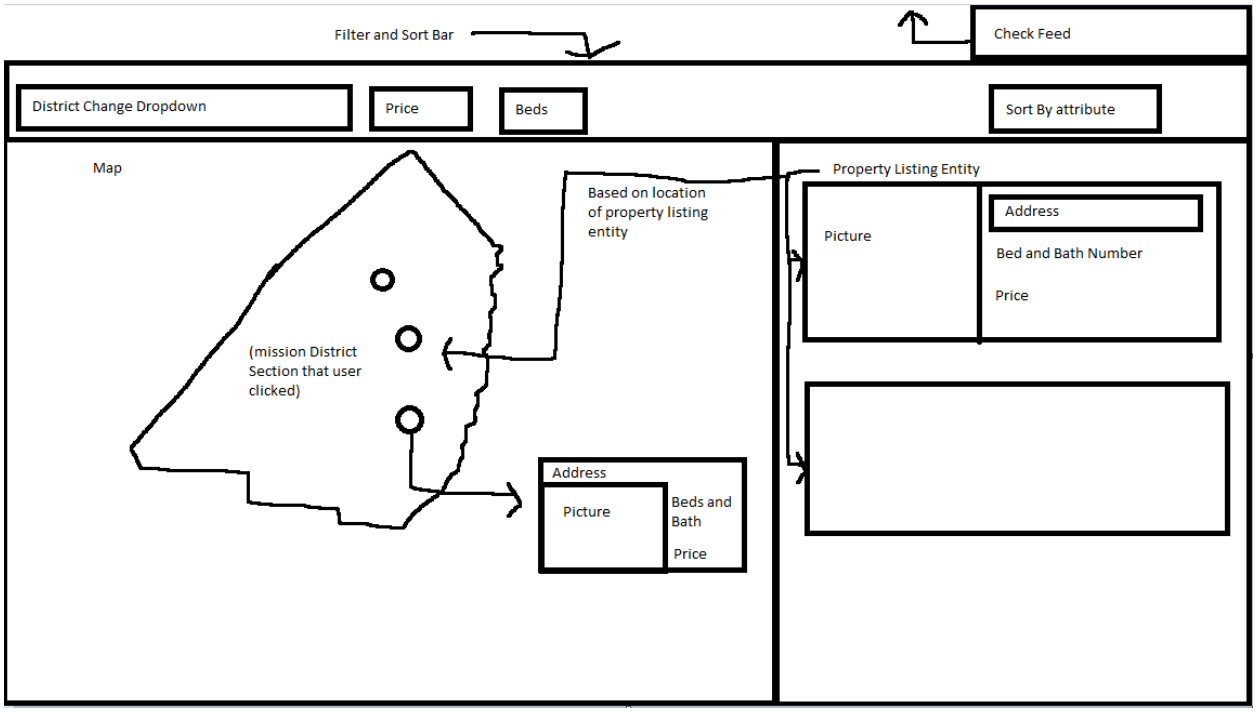
Social Page:



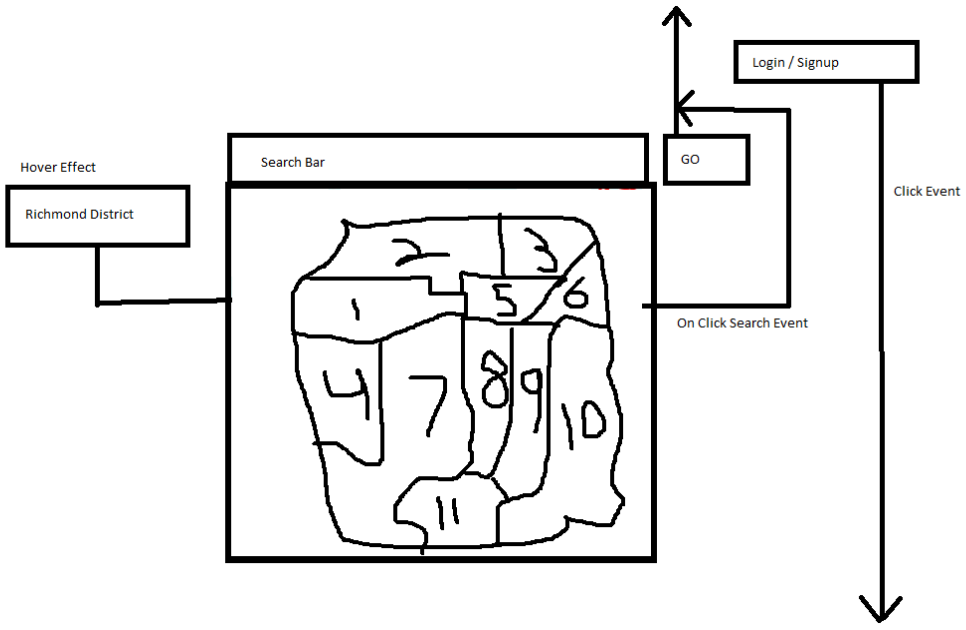
Messages:



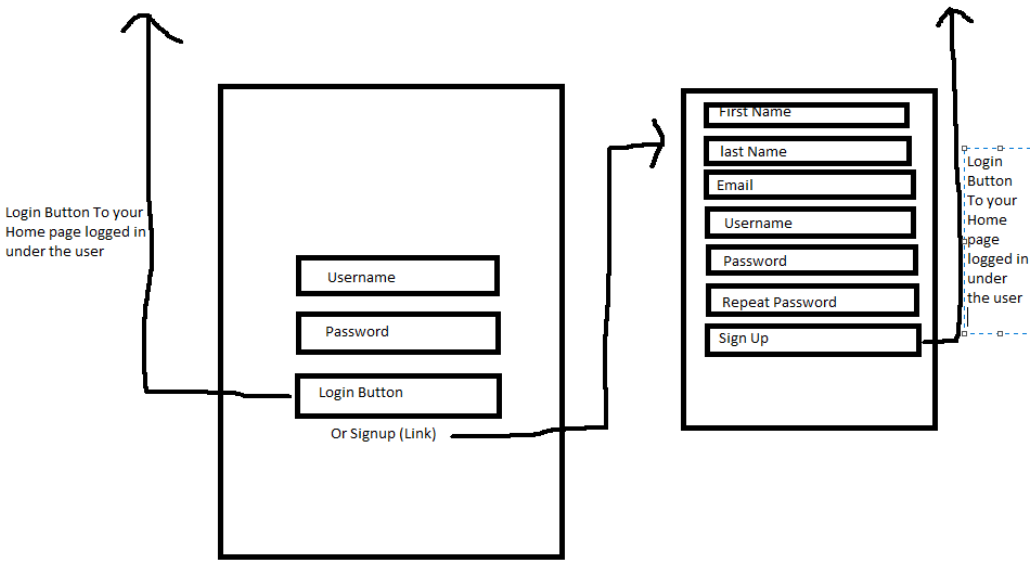
Property Listing:



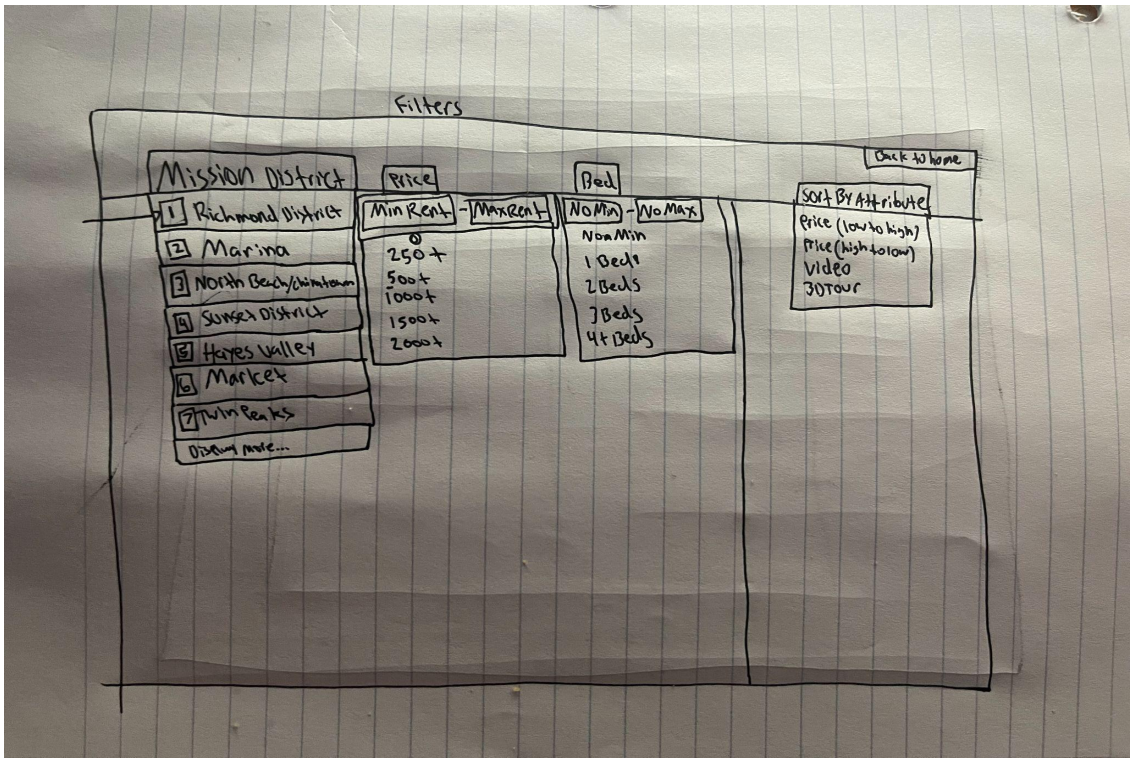
Main Search:



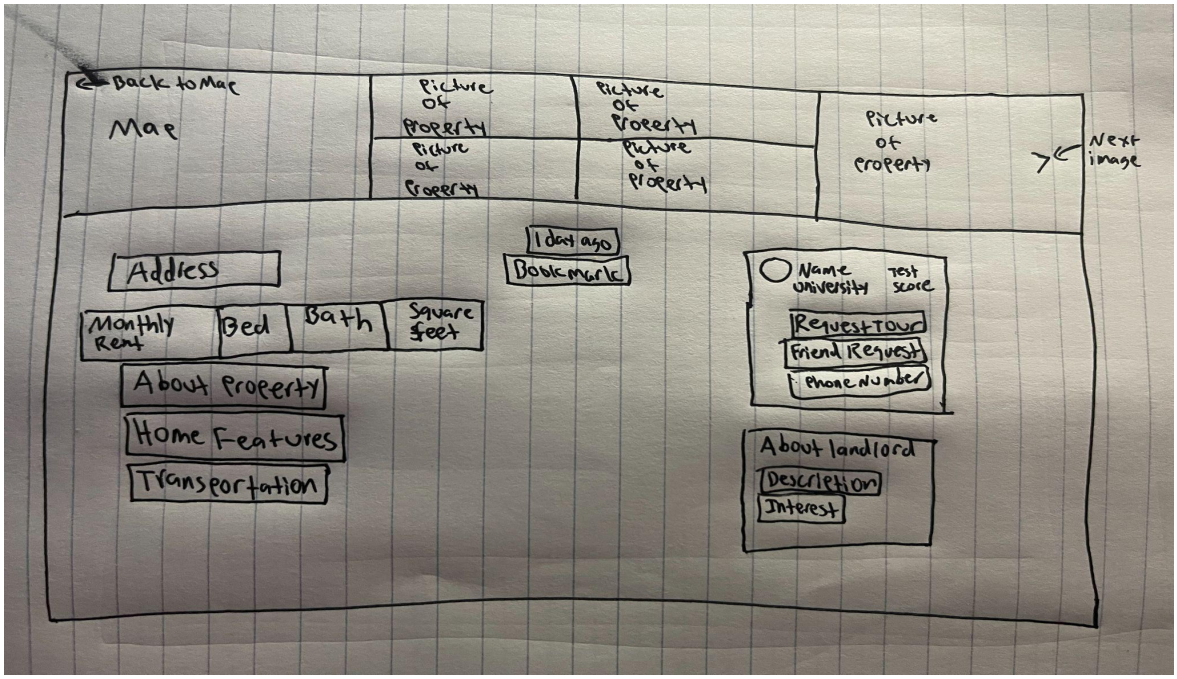
Login/Sign up Page:


Login Button To your Home page logged in under the user
Username
Password
Login Button
Or Signup (Link)
First Name
last Name
Email
Username
Password
Repeat Password
Sign Up
Login Button To your Home page logged in under the user

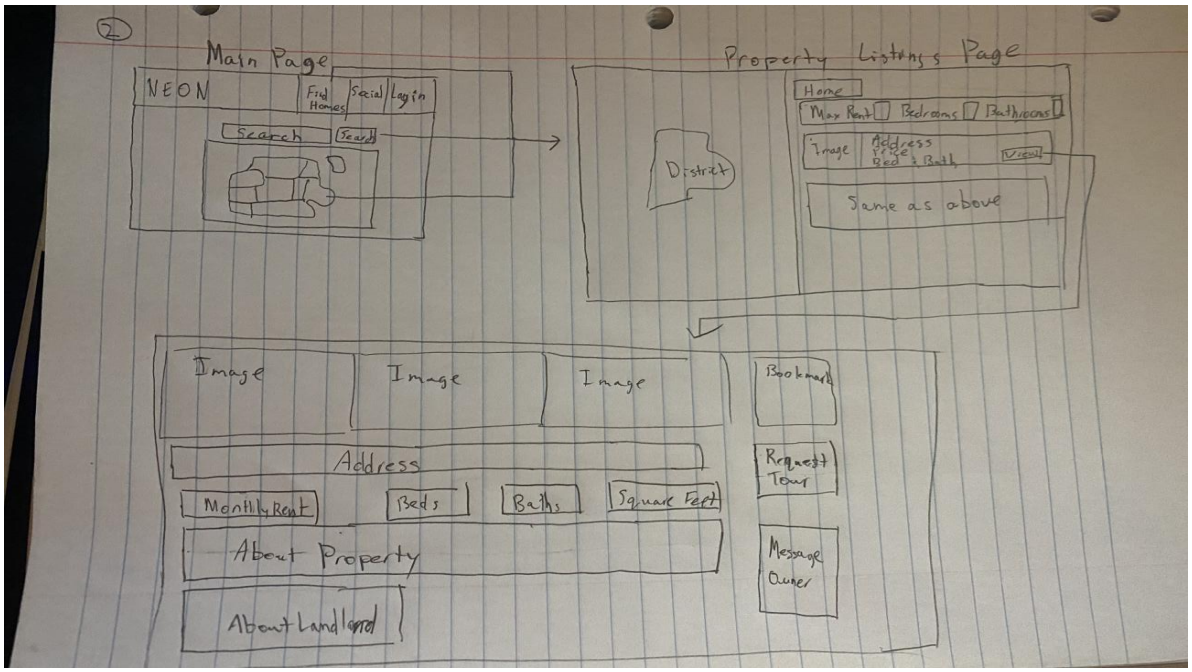
Filters:

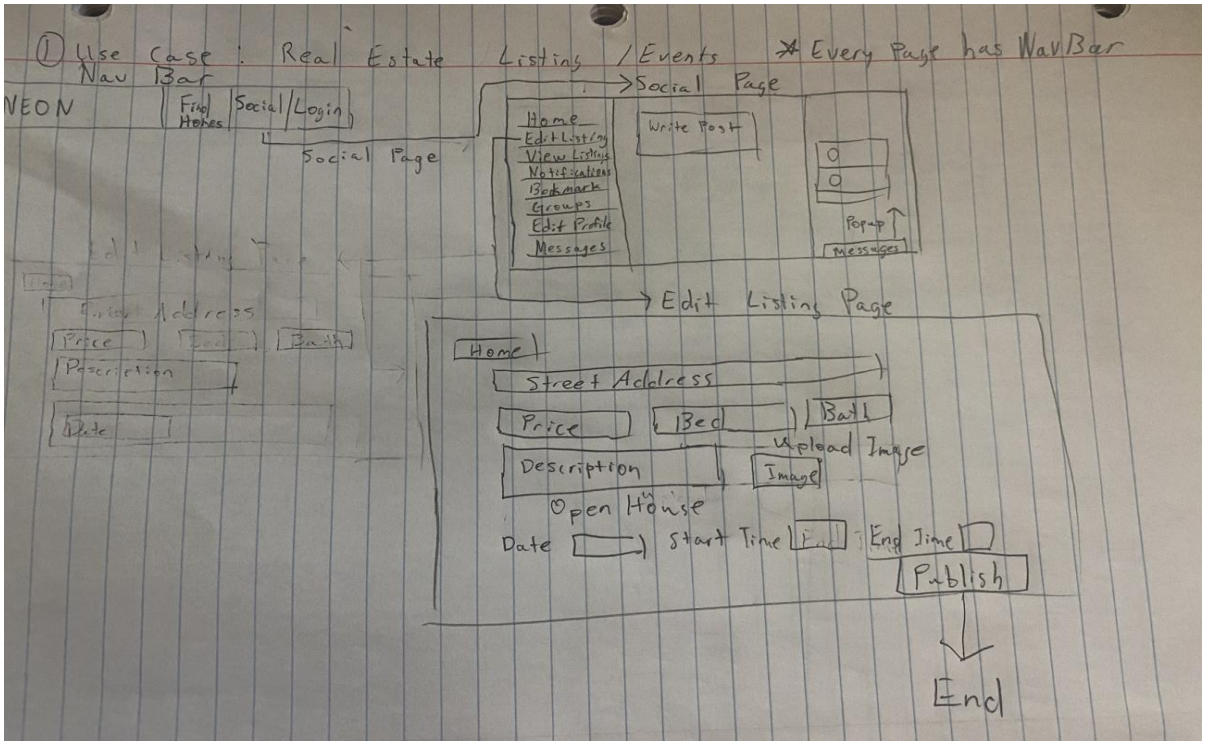

Filters
Mission District
1 Richmond District
2 Marina
3 North Beach/Chinatown
4 Sunset District
5 Hayes Valley
6 Market
7 Twin Peaks
Price
Min Rent - MaxRent
250+
500+
1000+
1500+
2000+
Bed
No Min - No Max
Non Min
1 Beds
2 Beds
3 Beds
4+ Beds
Back to home
Sort By Attribute
Price (low to high)
Price (high to low)
Video
3D Tour

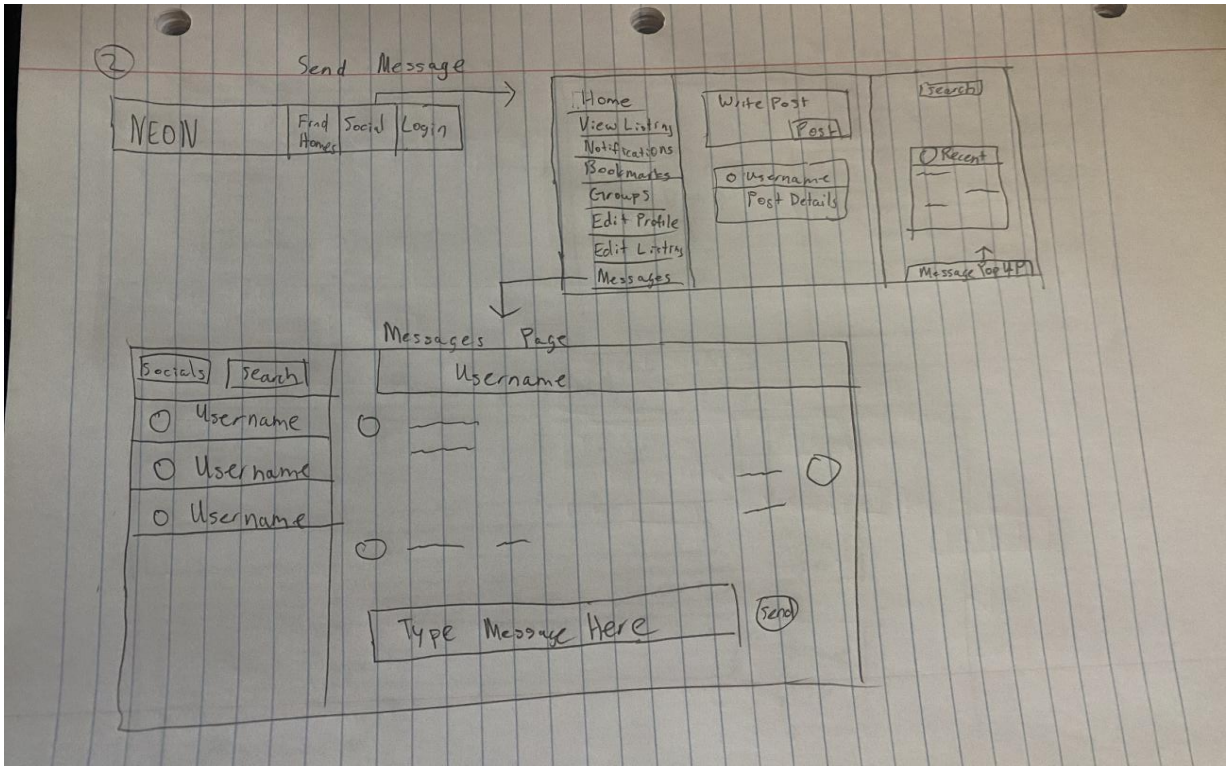
Displaying Property Listing:

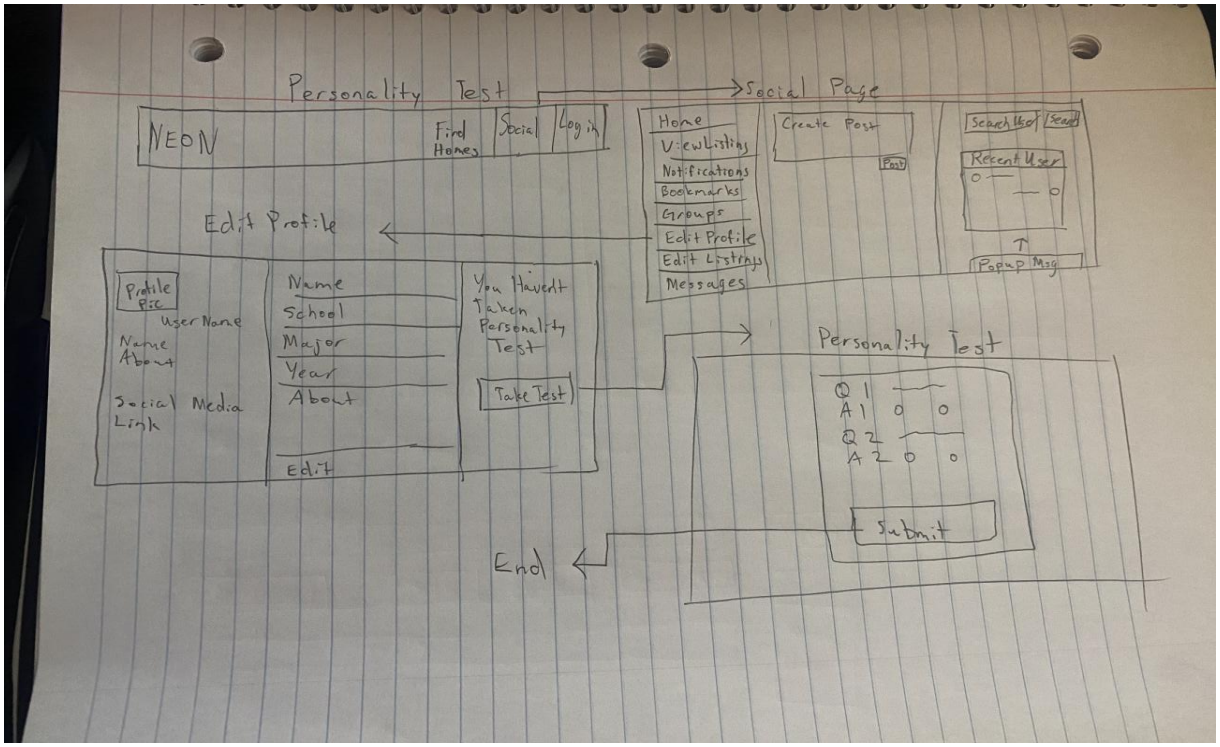


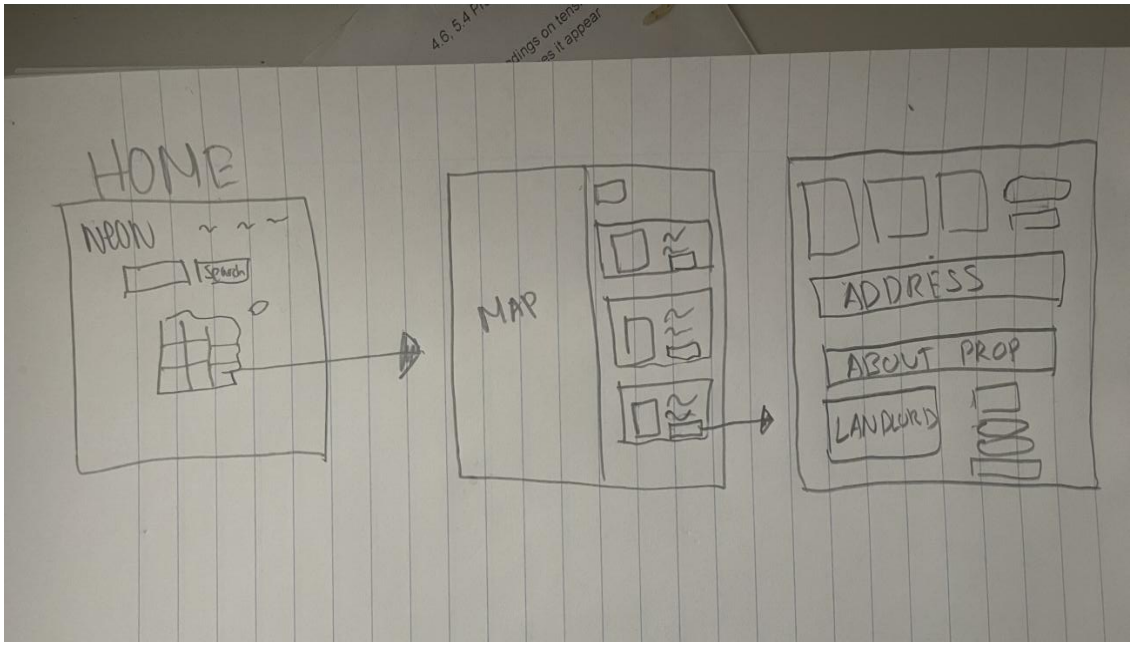
Storyboards:

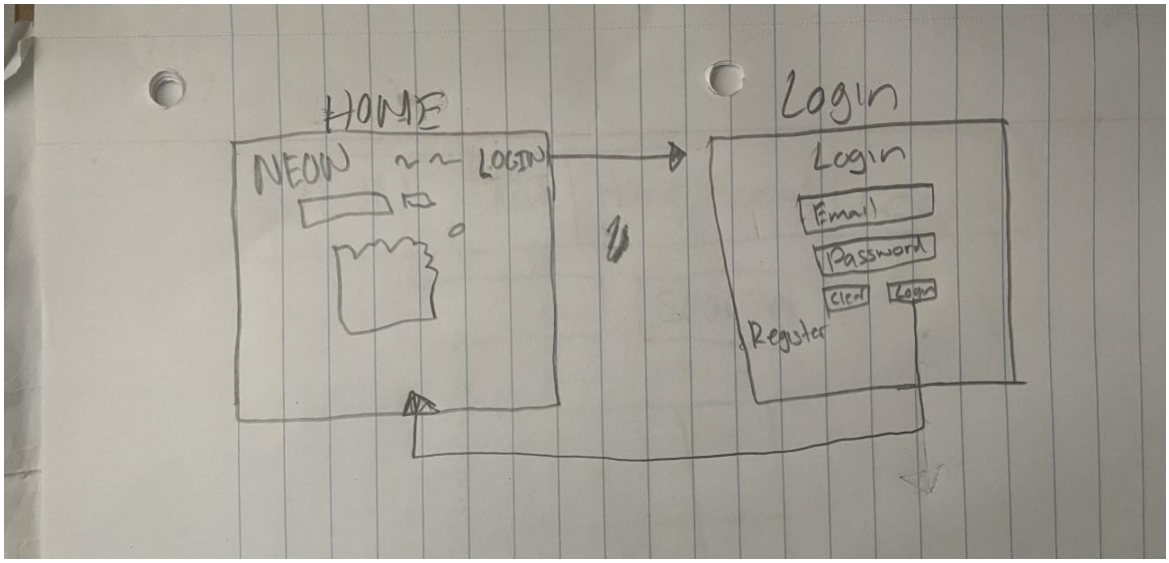

2
Main Page
NEON
Find Homes
Social/Login
Search
Search
Property Listings Page
Home
Max Rent
Bedrooms
Bathrooms
District
Image
Address
Price
Bed
Bath
View
Same as above
Image
Image
Image
Bookmark
Address
Request Tour
Monthly Rent
Beds
Baths
Square Feet
About Property
Message Owner
About Landlord

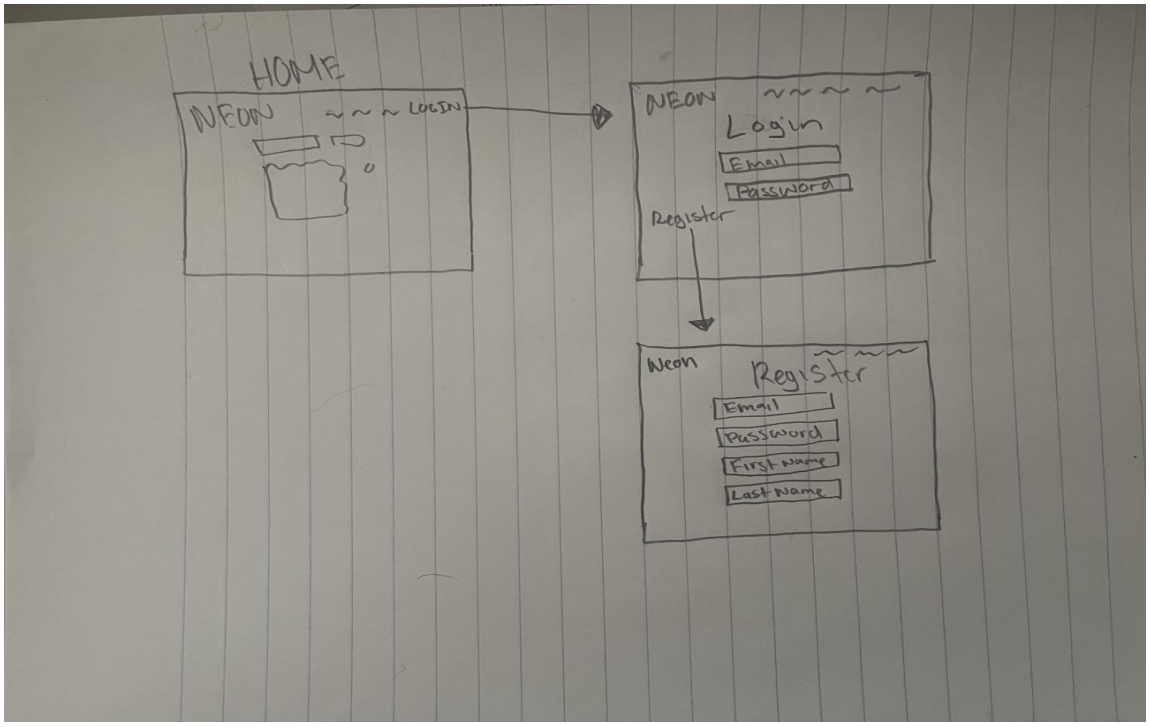

1 Use Case: Real Estate Listing / Events
* Every Page has NavBar
Nav Bar
NEON
Find Homes
Social/Login
Social Page
Social Page
Home
Edit Listing
View Listings
Notifications
Bookmark
Groups
Edit Profile
Messages
Write Post
Pop-up
Messages
Edit Listing Page
Home
Street Address
Price
Bed
Bath
Description
Upload Image
Image
Open House
Date
Start Time
End Time
Publish
End

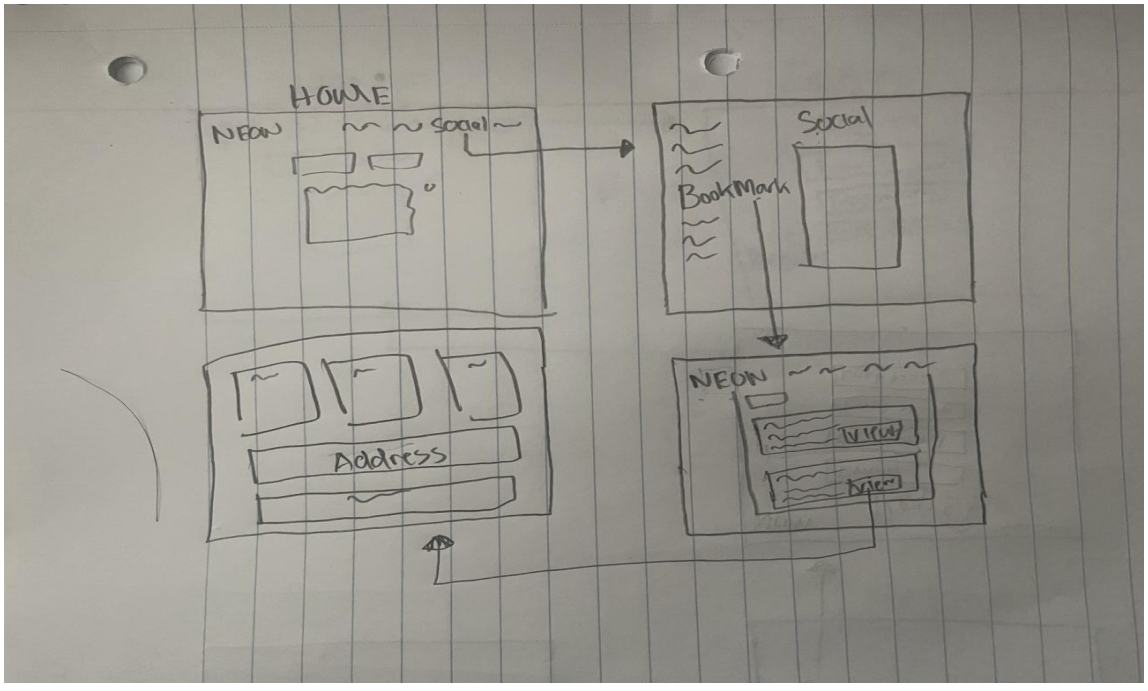

2
Send Message
NEON
Find Homes
Social
Login
Home
View Listing
Notifications
Bookmarks
Groups
Edit Profile
Edit Listing
Messages
Write Post
Post
Username
Post Details
Search
Recent
Message Pop UP
Messages Page
Socials
Search
Username
Username
Username
Username
Type Message Here
Send


Personality Test
Social Page
NEON
Find Homes
Social
Log in
Home
View Listing
Notifications
Bookmarks
Groups
Edit Profile
Edit Listings
Messages
Create Post
Post
Search User
Recent User
Popup Msg
Edit Profile
Profile Pic
User Name
Name
About
Social Media Link
Name
School
Major
Year
About
Edit
You Haven't Taken Personality Test
Take Test
Personality Test
Q 1
A 1
Q 2
A 2
Submit
End


HOME
NEON
Search
MAP
ADDRESS
ABOUT PROP
LANDLORD


HOME
NEON
LOGIN
Login
Login
Email
Password
Clear
Login
Register


HOME
NEON
LOGIN
NEON
Login
Email
Password
Register
Neon
Register
Email
Password
First Name
Last Name


HOME
NEON
Social
Social
BookMark
Address
NEON

4. High level database architecture and organization

1. Background Check:

- A background check shall have a unique ID.
- A background check shall be tied to a userID.
- A background check shall contain Employment Record of userID.
- A background check shall contain Education Record of userID.
- A background check shall contain Financial Record of userID.

2. Registered User:

- A registered user shall have one password
- A registered user shall have one username
- A registered user shall have one or more email on the account.
- A registered user shall have either be a renter or a landlord.

3. Location of Rental Listings:

- Each location shall have 1 and only 1 locationID.
- A location shall 1 and only 1 address
- A location shall 1 and only 1 GPS Coordinates
- A location shall 1 and only 1 Region ID

4. Region

- Each Region shall have 1 and only 1 RegionID
- Each Region shall have many LocationID's.
- Each Region shall have 1 and only 1 District

5. Rental Listings:

- One rental listing can have many posts.
- A rental listing shall have 0 or many beds
- A rental listing shall have 0 or many bathrooms
- A rental listing shall have one and only one address
- A rental listing shall have one and only one price

- A rental listing shall have one and only one property type
- A rental listing shall have many details

6. Posts:

- One post can have many comments.
- Each post shall have one and only one postID.
- Each post shall be tied one and only one userID.

7. Messages:

- Each message shall have one and only one FROM userID.
- Each message shall have one and only one TO userID.
- Each message shall have one and only one timestamp.
- Each message shall contain text(s).

8. Notification:

- Each notification shall be tied to one or more registered user.
- A notification shall be tied to a message or post.
- A notification shall contrarian one and only one time stamp.

9. Events:

- Each event shall have one and only one eventID.
- Each event shall have at least 1 registered user.
- Each event shall have one or more date and time.

10. Friend Request:

- Each friend request shall have one requestID.
- Each friend request shall have only one FROM userID.
- Each friend request shall have only one TO userID.
- Each friend request shall contain a time and date.

11. Saved Properties List:

- Each saved property shall have zero or many PropertyID.
- Each saved property shall have one and only one timestamp.
- Each saved property shall have one and only one userID.
- Each saved property shall have zero or many listingID

12. Groups:

- A group shall have many registered users.
- A group shall have zero or more posts.
- A group shall have one and only one groupID.

13. Reports:

- A report shall have one and only one reportID.
- A report shall be tied to a listingID.
- A report shall contain a time and date.

14. Comments:

- Comments shall have 1 and only one UserID.
- Comments shall have 1 and only one timestamp.
- Comments shall contain texts.

Entities, Attributes, Relationship, and Domains

1. **Registered User**
   - User_ID: Key, Alphanumeric
   - Password: Composite, Alphanumeric
   - BC_Report_ID: Key, Numeric
   - First_Name: Alphabetic
   - Last_Name: Alphabetic
   - Email: Composite, Alphanumeric

2. **Background Check Report(Unsure)**
   - Report_ID:Key, Numeric
   - User_ID:Key, Alphanumeric
   - Status: Composite, Alphabetic
   - Credit_Score: Numeric
   - Income: Numeric
   - Rental_History: Alphanumeric

3. **Location of Rental Listing**
   - Location_ID: Key, Numeric
   - Address: Composite, Alphanumeric

4. **Region**
   - Region_ID: Key, Alphanumeric
   - Region_name: Alphanumeric
   - Location_ID: Key, Numeric

5. **Rental Listing**
   - User_ID: Key, Alphanumeric
   - Listing_ID:Key, Numeric
   - Location_ID: Key, Numeric
   - Rooms: Numeric
   - Bathrooms: Numeric
   - Price: Numeric
   - Property Type: Composite, Alphabetic

6. **Posts**

-Posts_ID:Key,Numeric
- User_ID:key, Alphanumeric
- Comment_ID:Key,Numeric
- Post_Content: Alphanumeric

7. **Comments**
   - Comment_ID:Key,Numeric
   - User_ID:key, Alphanumeric
   - Text:composite, Alphanumeric
   - Date/timestamp: Multivalue, Timestamp

8. **Messages**
   - Message_ID:Key,Numeric
   - To_User_ID:Key, Alphanumeric
   - From_User_ID:Key, Alphanumeric
   - Text:composite, Alphanumeric
   - Date/timestamp: Multivalue, Timestamp

10. **Events**
    - Event_ID:Key,Numeric
    - Host_User_ID:Key, Alphanumeric
    - Event_Name: Alphanumeric

11. **Event Guest**
    - Event_Guest_ID:Key, Alphanumeric
    - User_ID:Key, Alphanumeric
    - Event_ID:Key,Numeric

12. **Friend Request**
    - Request_ID: Key,Numeric
    - To_User_ID: Key, Alphanumeric
    - From_User_ID: Key, Alphanumeric

13. **Friend Request List**
    -Request_List_ID: Key,Numeric
    -To_User_ID:key, Alphanumeric
    -From_User_ID: Key, Alphanumeric

14. **Friend List**
    -Friend_List_ID: Key,Numeric

-Friend_ID: Key, Alphanumeric
-User_ID:Key, Alphanumeric

15. Saved Properties

-Saved_Properties_ID: Key,Numeric
- User_ID: Key, Alphanumeric
- Listing_ID: Key, Alphanumeric

16. Groups

-Group_ID: Key,Numeric
-Host_User_ID: Key, Alphanumeric
-Group_Name: Alphanumeric

17. **Group Guests**

- Group_Guest_ID: Key,Numeric
- Group_ID: Key,Numeric
- Guest_User_ID: Key, Alphanumeric

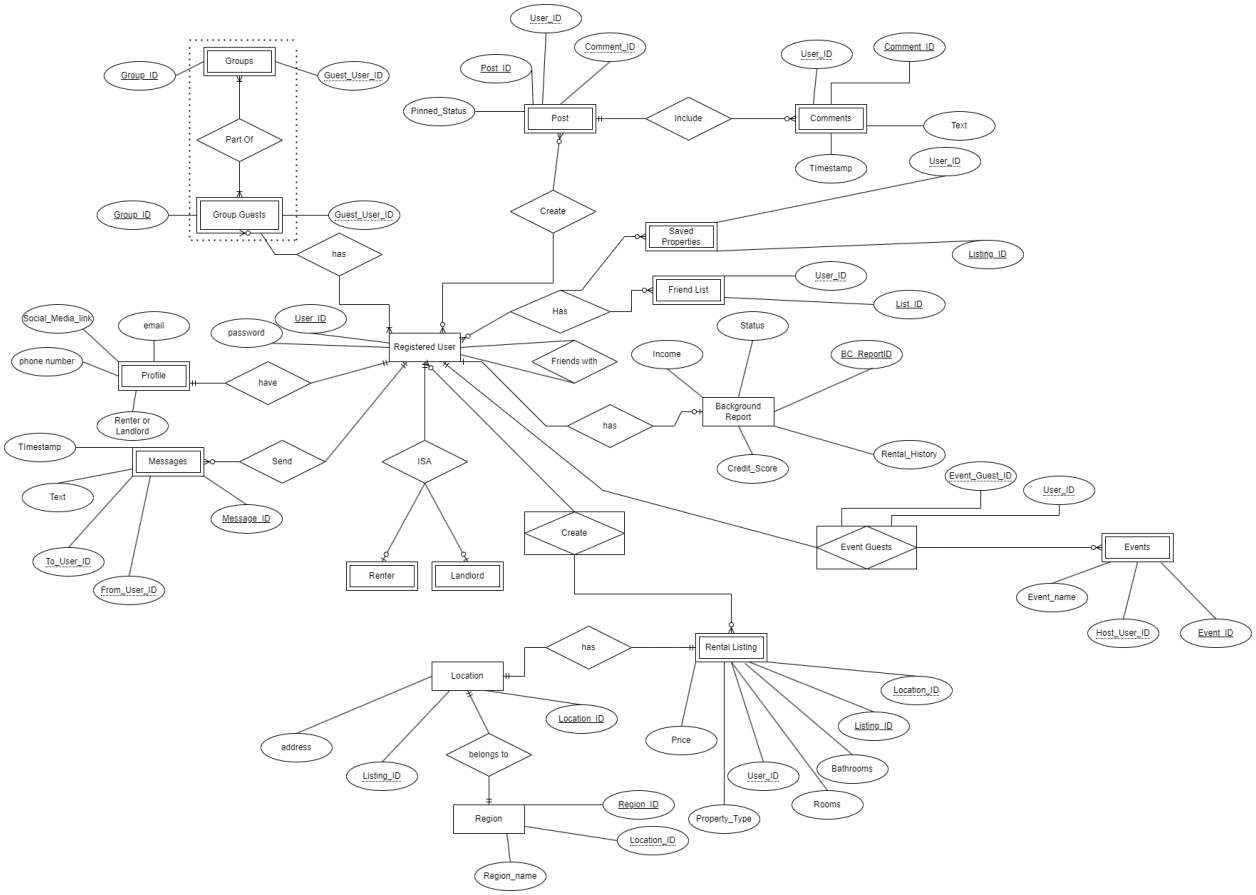
18. Reports

-Report_ID: Key,Numeric
-User_ID: Key, Alphanumeric
-Report_INFO: Alphanumeric

19. Profile

- Profile_ID: Key,Numeric
- User_ID: Key, Alphanumeric
- Phone number: Numeric
- Social_Media_Link: Alphanumeric
- Account_Type: Alpha

Entity Relationship Diagram (ERD)



- We chose MySql because it is a relational database management system that is widely known and adopted in many different fields.

Media storage

The application requires two media storage types: images for each listing and GPS locations to use with Google Maps API. To handle images, they will be stored in a file system since blobs within the MySQL database would significantly impact performance. To handle images, an initial directory will be created with subdirectories for each district. Then, each time the user makes a new listing with images, the API will generate a new folder within the district directory in which the listing is located. The folder name will

correlate to the address with a set number of characters to prevent exceeding the limit of the directory path attribute stored within the database. After the directory is created, the images will be stored in the folder, and the folder path will be stored in the images attribute of the listing entity. Once complete, the API will be able to use the folder path from the images attribute to retrieve the images for that listing. To handle the GPS location for each listing, the MySQL POINT type will be used. When the user creates a new listing, the Google Maps API will verify the address, and the longitude and latitude values will be stored in the location POINT for the listing entity. Then, when the user requests a listing, the POINT values can be used to view the location within Google Maps.

Search/filter architecture and implementation

To handle searching and filtering for listings, a MySQL query builder function will be created based on the user's requirements for district, price range, and number of rooms. The function will filter listings from the selected Region with a price range between the chosen lower and upper bounds and the required number of rooms. The sort attribute will then sort the resulting listings and return a maximum of 10 listings.

5. High Level APIs and Main Algorithms

User authentication API (Priority 1):

1. Creation of accounts:

This API allows users to create accounts and register on our platform. Includes steps:
The Unregistered user inputs their account information like username, email, password, repeat password, and etc.
Client side validation ensures that the data is correctly formatted.
When the user clicks the register button, a POST request is sent to the Node API server.
Server-side validation checks if the provided data is valid.
If everything is valid, the user's information is added to the database.
Upon successful registration, the user is redirected to the login screen.

2. Logging into accounts:

This API is for user login:
Users input their username and password.
A POST request is sent to the server to check if the user exists in the database.
If the username is validated, the API creates a session for the user, allowing them to remain authenticated while using the platform. Then the user is redirected to their Home page.
The API also handles errors effectively, providing meaningful feedback to users to resolve any login issues. If the user enters an incorrect username or password, the API will display an error message to the user.

3. Background Checking (Priority 1):

This API will retrieve data about any potential tenant or landlord using existing. APIs to search for any misdemeanors, court records, driver history,rental history, income, and credit history. Once the background checking process has concluded a checkmark symbol will appear next to an registered user’s account denoting that they are background checked.

When a landlord triggers a background check, the API should initiate a series of verification processes.
This might include sending requests to external credit agencies and databases to cross-reference the provided information.
The API should handle the responses from these external services and maintain logs of these interactions.

4. Group Notification and Saved Properties API (Priority 2):
This API is responsible for managing group notifications and saved properties within our platform.
Users can receive and view group notifications related to properties they are interested in.
Additionally, users have the option to save properties for future reference.
The API handles the storage and retrieval of these notifications and saved property data
Implement authorization and authentication checks to ensure that only authenticated users can access their notifications.
Query the database for notifications associated with the user's account and return the data in a structured format, typically JSON.

5. Quality of Client API (Priority 3):
Implement code to continuously monitor key performance metrics, such as page load times and responsiveness, in real-time.
Set up automated systems that detect performance issues in real-time.
Use alerting mechanisms, such as email, or webhook notifications, to immediately inform administrators when issues are detected.
Store detailed issue information in logs or a database for further analysis.

Data retrieval API (Priority 1):

1. **Listing and retrieving rental listings Information:**

This API handles the creation of new rental listings, ensuring that all required fields are filled in. It also retrieves rental listings information, including title, description, image, price, number of bed and bath, and address.

Create Listings: This API will handle validation to ensure that the provided listing information is complete and valid, including details like property type, location, price, number of bed and bath, image, title, property description.
Store the listing information in a database, associating it with the user who created the listing.
Retrieve Listings: This API will handle query parameters that allow users to filter listings based on criteria such as location, price range, and property type.
Query the database to fetch relevant listings based on the specified filters and return the results in a structured format, such as JSON.
Update Listings: This API will handle validation to ensure that only the listing owner can modify their own listings.
Update the listing information in the database based on the provided changes.
Delete Listings: This API will handle authorization checks to confirm that only the listing owner can delete their listings.
Remove the listing from the database, ensuring it's no longer accessible.

Search and recommendation API (Priority 1):

2. **Search and recommendation API:**

   Property Search: This API will handle search queries and generate personalized recommendations based on user preferences and behavior. The purpose is to enhance user engagement and content discovery within the platform

   This API will handle query parameters for search criteria such as location, property type, price range, and other relevant filters.
   Query the database to fetch properties that match the specified criteria and return the results in a structured format, such as JSON.

   Recommendations: This API will handle recommendation algorithms or models based on user behavior, preferences, and historical data.
   This API uses these algorithms to suggest properties that are likely to match the user's interests and return the recommendations in a structured format.

Notification and Messaging API (Priority 2):

3. Notification and Messaging:

This API will manage notifications, messages, and alerts sent to users. It maintains an array of messages, with each message containing the ID of the message poster, the message itself, and the timestamp. The purpose is to enable real-time communication and keep users informed about relevant activities.
This API will handle validation to ensure that the message content is provided, the recipient is valid, and the sender has the necessary permissions.
Also, it stores the messages in the database, associating them with sender and recipient information.

4. Filtering of Listing by Location API (Priority 1):

This API will manage our listing based on the user's location filtering preferences. It will show all the listings within a specific geographic area. The purpose is to ensure the user has an option to filter out her listing giving them the results that aligns with their specific location criteria.
This API will handle parameters for location-based filtering, such as city, neighborhood.

5. Detecting Inappropriate Posts or Accounts (Priority 1):

This API will detect and handle reports and the inappropriate content that is being posted on the app. The purpose is to ensure the user can browse through safe and suitable content that is free from scam and other fraudulent activities.

User Reporting: This API will handle validation to ensure that the reported content or account is properly identified and reported for legitimate reasons.
Image Analysis: This API will recognize and analyze reported images to identify inappropriate content.
Blocking or Flagging: This API will develop a mechanism to flag or block inappropriate content or accounts. It allows administrators to review flagged content and take appropriate action, such as removing content or suspending accounts.

6. API to Link Social Media Accounts such as Twitter or Instagram (Priority1):

This API will allow users to link their social media accounts to their SFStudentRent account. The purpose of linking social media accounts is to enhance the app's security by preventing fraudulent accounts and enabling the users to find potential roommates.
This API will handle authentication protocols to allow users to authorize access to their social media accounts securely. Once authorized, associate the linked social media account with the user's account on your platform. It stores the association in the database, mapping the user's account to their social media account identifiers.

Non-trivial algorithms or processes:

1. Housing Price Forecast Calculation (Priority 2):

This process will forecast possible changes in rent or housing prices for the upcoming years using existing APIs and public data available based on the zipcode of a property. It will take a variety of factors and retrieve data for other properties in the same area, look at an area jobs' outlook and schooling to come up with an accurate prediction of how a property forecasts to increase in price or rent.

2. Deep Learning/AI Housing Forecast Calculation (Priority 3):

This process will attempt to predict the same rent or housing prices but for a longer period such as the upcoming ~5 to 20 years and will do so by using machine learning algorithms and will use existing APIs in the beginning to create its own API to model the regression or progression that it expects a property to have for a given time interval. The model will get smarter and better with more data as it gets fine-tuned and the results of its predictions come in. This will be especially useful for home buyers on deciding what the best area to buy a property will be.
This API will define input parameters, including property details such as location, type, historical pricing data, and forecast period. This API will Integrate linear regression, time series analysis, or machine learning models for price forecasting.

Posts and Messages Integration API:

1. Real-time Chat Synchronization Process (Priority 2):
This API will make store all the messages that take place as an array of objects, with each object having its own message ID, ID of the message poster, and timestamp. This process ensures that messages sent between users in real-time chats are synchronized across devices.The purpose is to provide a seamless messaging experience for users.

2. Promotion or Pinned Post API (Priority 2):
This API will first identify the content that is desired to be promoted or pinned to an user account and it will strategically push the listings to applicable users.

Promoted listing will have a changed priority at the same level as a brand new post and will have a greater change at showing up in a user's recommendations.

The pinned posts are subject to the user's preferences with a limit of three per account and promoted listing are intialized by landlords who wish to get a larger reach for their property or sell their property quicker with a limit of two free promotions a week.

3. Editing or commenting Post API (Priority 1):
This API will store each post in an array of objects having its own post ID, ID of the user who created Post, and timestamp of post. Posts can either come in the form of landlord posting rental listings, or renters posting looking for roomates.

If any changes are made by a user to update his or her post, the API will retrieve and change the current data of the post. In addition it also allows users the functionality to edit their own post, comment, replies storing each of the comments in its own array with comment ID, ID of commenter, and a timestamp. The poster has capability of allowing comments for all users or only registered users or only friends.

All comments will go through the the Spam Messages API and are subject to be deleted for highlighted for an account to be banned if the API detects any unusual or uncanny activity.

4. Detecting Inappropriate Messages or Spam Messages API (Priority 2):

This API will first make a call to see if an account has an IP address, email, or domain that is on a blacklist and if it detects that it does it will block the account which will be very useful in preventing against spam or bot regristrations.

Any future accounts that get banned will be added to the blacklist and any users with the blocked IP will be unable to use the service.

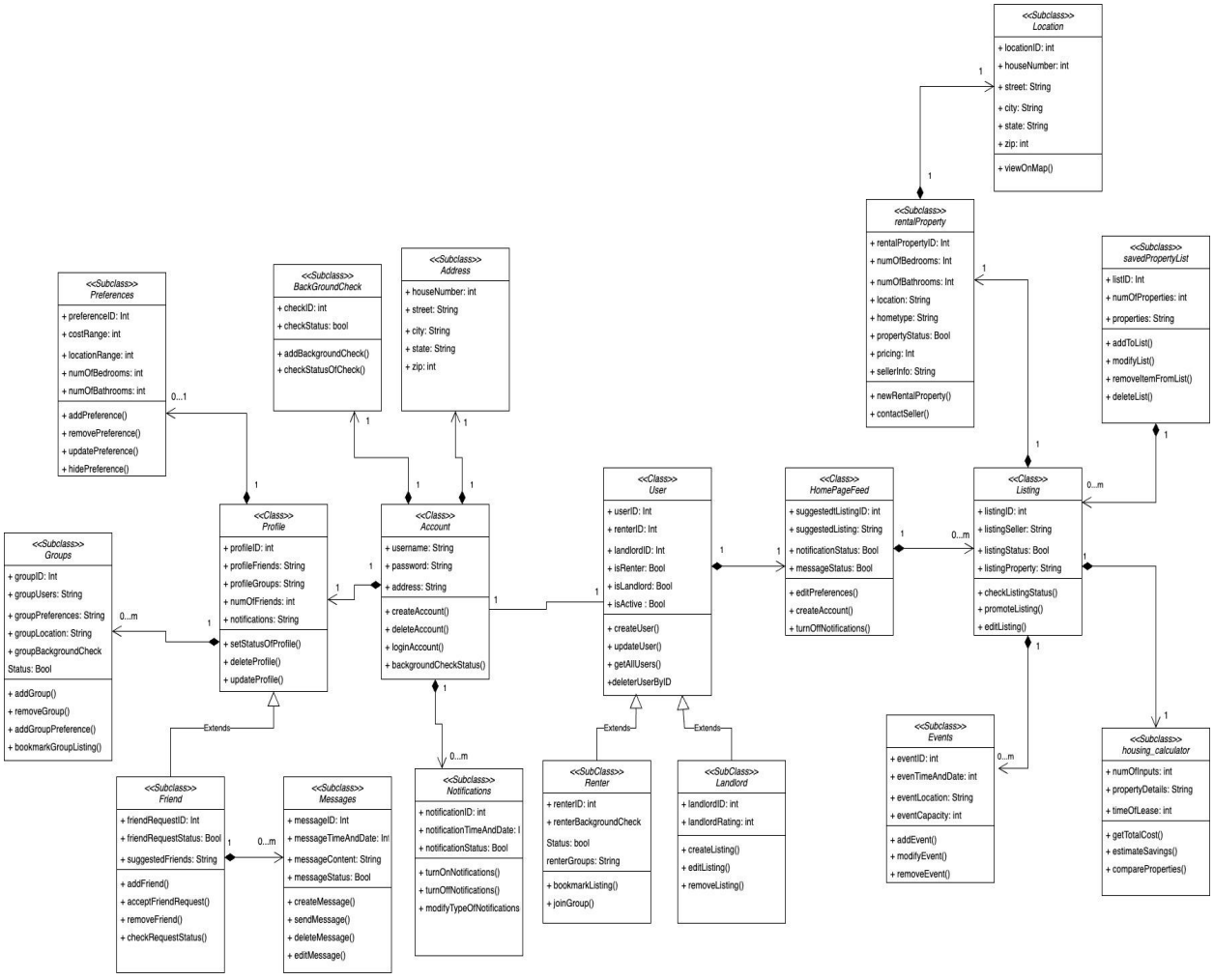
There will also be an API call to scan for inappropriate words or phrases and immediately be highlighted and subject to review by administrators who will decide what to do with the in question account.

Finally there will be an interface for reporting all misdemeanors and inappropriate actions of another user with administrators being the final step in determining the status of whether or not to ban and blacklist an account.

6. High Level UML Diagrams

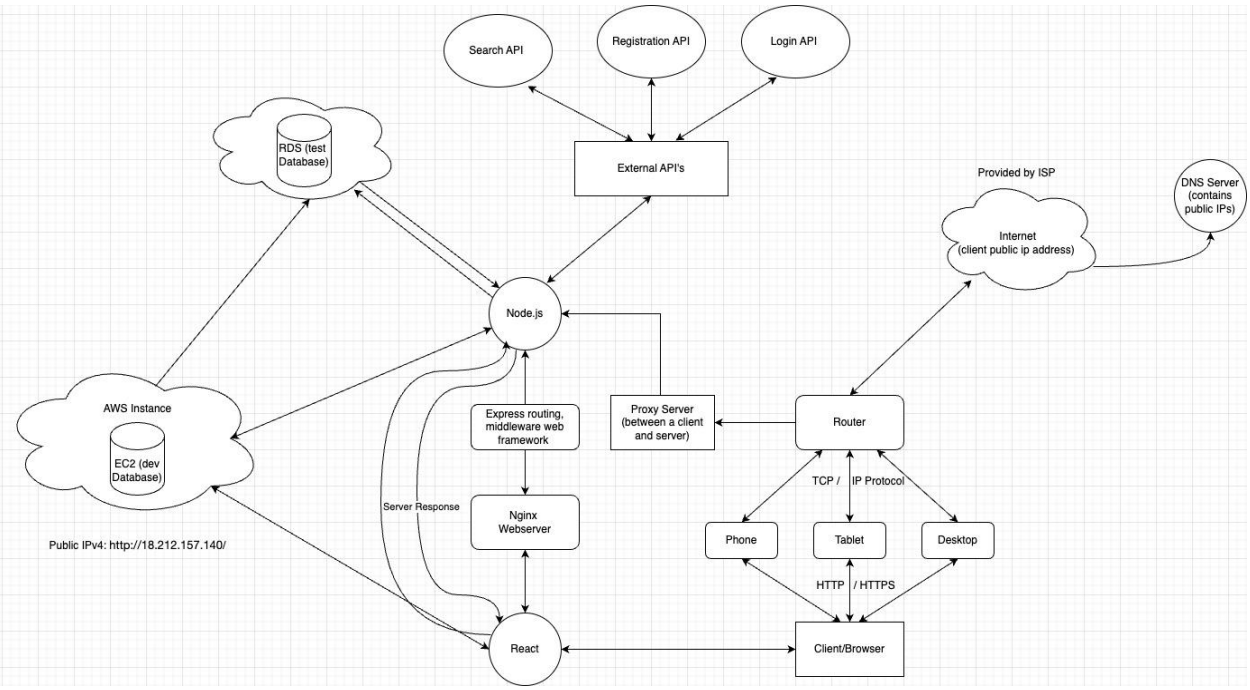
• High-level UML class diagrams for implementation classes of core functionality, i.e. functionality with provided interfaces. Focus on a main high-level classes only (one or at most two levels deep). Your UML diagram must be implemented using Object Oriented or Protocol Oriented approaches as seen in Class.

• Use data terms and names consistently with Data Definition Section 1 above.
https://drive.google.com/file/d/1OCo-y7JNa5F4_hOWmugzJgZMKJdR4jFo/view?usp=sharing

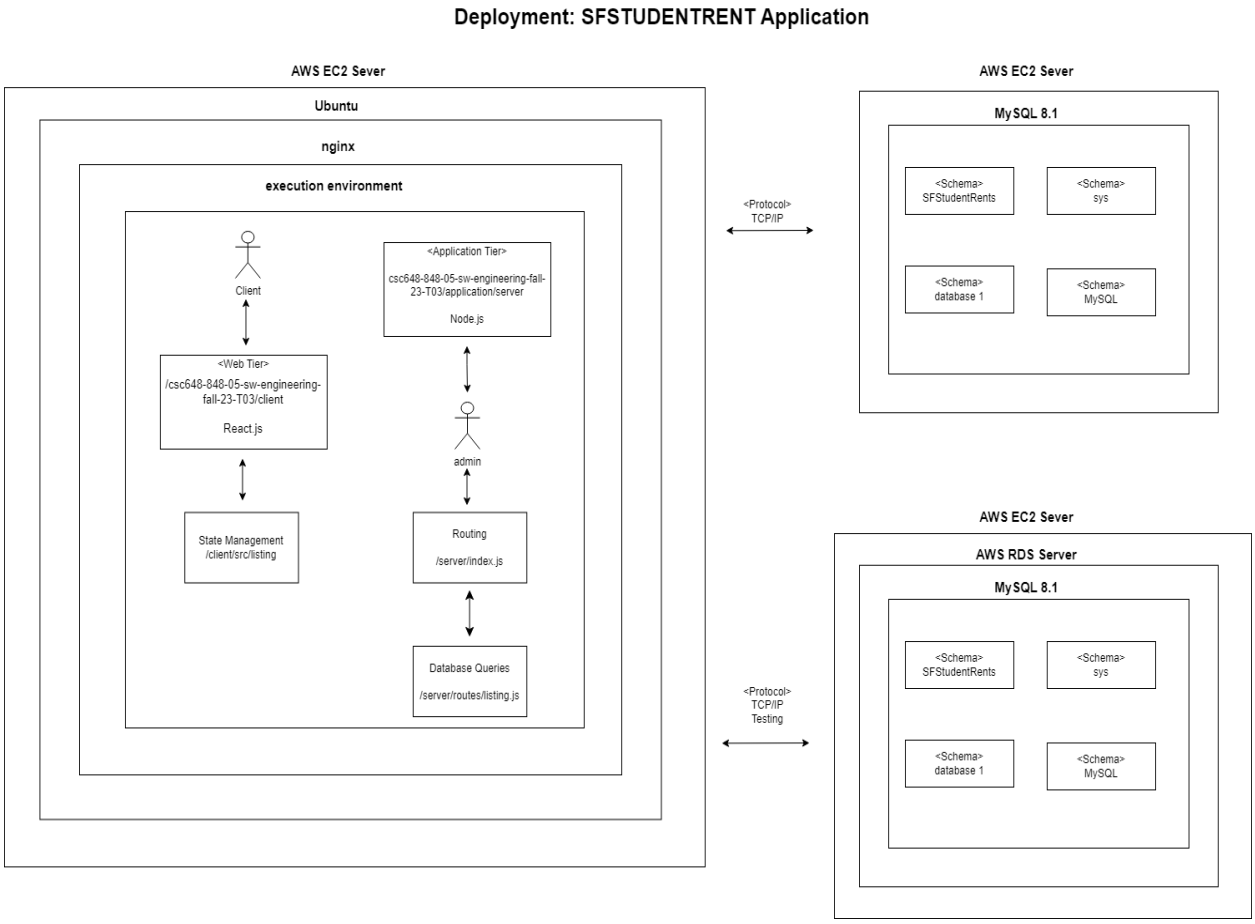


7. High Level Application Network and Deployment Diagrams

Application Network Diagram:
https://drive.google.com/file/d/1UWdu-Du5dFywLpYWD4owqKvRh9_iZX-9/view?usp=sharing



Deployment Diagram:
https://drive.google.com/file/d/1uRXj8tXfLRSPx-sQbFEYQnTkgl7Q9_LZ/view?usp=sharing



8. Key Risks

• **Schedule Risks:**

- A problem we noticed was that the top down project cannot begin until the database has something to work with, for example we cannot create our data hook api to connect to the front end. A way to resolve that is to ensure we have things done on time.

- We have a 7th member now and we can not find a time where we can all meet up. Since he is placed at the frontend time, we have decided to leave it to the frontend lead to fill him in on information he missed and the work he needs done.

• **Legal Risks:**

- Potential problem with certain discrimination housing laws in CA, discrimination on criminal history. Felons are ok to, but petty misdemeanors for example may or may not necessarily warrant certain limitations. Way around this is maybe a minimum standard contract, something like that, where homeowners can put a minimum standard for their tennant, but they can probably more easily relate that to a different protected class which makes it a very grey area. A way to resolve this is the work around stated or focus on this part later to get the proper research on the legalities.

•**Technical Risks:**

- Our team of eight members and four of us want to go into the frontend team. Our backend and database team only has two people there. We plan on resolving this by shifting people around to rebalance the teams and the team lead plans to go to the team that needs the most support.

- The team is working with Amazon AWS using the EC2 instance to host our server and RDS to host our website. With the way Amazon has their cloud services set up, the team runs into error accessing certain part of the application. This could be resolved by starting the work early and finding the roadblocks before the due date gets too near.

9. Project management:

• So far we have been using discord to chat and manage our tasks, with individual calls and split each other into teams. We have been coordinating our work through Google Docs by leaving suggestions and comments for the next person to see. Currently we are now using Trello to manage each task in a more organized fashion that offers a unified dashboard view of all tasks and statuses and as well as who is assigned to each task. We also are using the notes and comments section in Trello cards to add further details to each task to allow us to manage our project properly. We also have color coordination on our Trello so the team knows what is being worked on, not started and what is finished. Currently this is doing a decent job to manage each task, thus we will probably continue to use this in the future as our main managerial tool. We also now have the ability to pin chat messages in discord, and that has been useful for our important links and feedback in that channel.

10. Detailed list of contributions

Members	Contributions	Score
Jeremy Tran	● Added data definitions for M2 ● Prioritized functional requirement ● Worked on Database Architecture ○ Worked on Database requirement and data definition, schema ○ Revised ERD ● Learned and left comments for revision on each section of M2 ● Revised M1 document based on feedback for V2 ● Created and invited team to Trello ● Checked and edited M1 and M2 document for consistency	
Geovanni Valadez	● Revised M1 document based on feedback for V2 ● Did data definition for M2 ● Prioritized functional requirements ● Created UI mockups and storyboard ● Lead frontend development of prototype ○ Split work among Ivan and Daniel ● Attended every meeting and contributed in discussions ● Has been on top of internal due dates	10
Mozhgan Ahsant	● Added data definitions for M2 ● Researched and planned out APIs needed for project ● Wrote Description of section 5 ● Has attended every meeting ● Finished work by due date	8
Anthony Silva	● Added data definition for M2 ● Attended every meeting and has contributed to discussions ● Worked on Database Architecture ○ Worked on database requirements, data definitions ○ Created ERD and schema	10
Aman Khera	● Added data definition for M2	6

	● Assisted in finding API and algorithms ● Created rough UML diagram and transferred to draw.io for production diagram ● Created rough Netwok and Deployment Diagram ● Has attended every meeting and contributed to group discussion ● Lead backend development of prototype	
Ivan Ayala-Brito	● Added data definition for M2 ● Organized M1 and M2 document ● Created UI Mockups and Storyboards ● Worked on frontend development of prototype ● Has attended every meeting	7
Daniel Enriqurez	● Worked on frontend development ○ Login Page ○ Register Page	10
Alex Huang	● Was proactive on getting caught up with the team ● Organized the document	10

Milestone 3

1. Data Definitions

1.User

- Unregistered: Limited to view-only access.
- Key Attitributes: N/A
- Relationships: None

2. Background Check

- Assesses registered user's financial and rental history.
- Key Attributes: Check_ID, Status
- Relationships: Linked to a registered user via Check_ID.

3. Registered User

- Full site access with functionalities like posting, messaging and joining groups.
- Key Attributes: User_ID, Password, Check_ID
- Relationships: Can have multiple rental listings, posts, messages, notification, events and linked to a background check.

4. Location of Rental Listings

- Precise location for rental listings.
- Key Attributes: Location_ID, Listing_ID
- Relationships: Linked to Rental Listings via Listing_ID.

5. Region

- General location of the rental listing such as neighborhood.
- Key Attributes:Region_ID, Location_ID
- Relationships: Contains multiple Locations

6. Rental Listings

- A post that contains media, images or videos along with information regarding the property.
- Key Attributes: Listing_ID, User_ID
- Relationship: Belongs to a registered user.

7. Posts

- Generic posts with or without media.
- Key Attributes: Post_ID, User_ID, Comment_ID

- Relationships: Owned by a registered user and can have multiple Comments.

8. Messages

- Private communication between registered users.
- Key Attributes: Message_ID, User_ID(to), User_ID(from)
- Relationships: Between two registered users.

9. Notification

- Updates for users about thei account activities.
- Key attributes: Notification_ID, Notification_Type
- Relationships: Received by Registered User.

10. Events

- User-created events either open or by invitation.
- Key attributes: Event_ID, User_ID(Host)
- Relationships: Hosted by a Registered User and contains other Registered Users as event guests.

11. Friend Request

- Registered users can add other registered users to their friend list.
- Key Attributes: Request_ID, User_ID(to), User_ID(from)
- Relationships: Between two Registered Users.

12. Saved Properties List

- Registered users can save preferred rental listings.
- Key Attributes: List_ID, Saved_ID
- Relationships: Owned by a Registered User.

13. Housing Cost Calculator

- Provides rental/buying suggestions based on input.
- Key Attributes: N/A
- Relationships: None(data is not saved)

14. Groups

- For group communications and activities.
- Key Attributes: Group_ID, User_ID
- Relationships: Owned by a Registered User and can contain other Register Users.

15. Reports

- Allows for users to report rental listings.
- Key Attributes: Report_Id, User_ID
- Relationships: Linked to a Rental Listing.

2. Functional Requirements

Priority 1:

1. **User:**

- (1.1) Users shall be able to register an account using a unique email and set up a password.

- (1.2) Users shall be able to browse the site without an account but will not be able to interact with posts or other users.

- (1.3) A user can reset their password if forgotten.

- (1.4) The system should verify user identities securely through email or phone number verification.

2. **Background Check Integration**

- (1.5) All registered users that wish to rent or rent out their place shall be subject to a background check.

- (1.6) The system shall integrate with predefined screening criteria and business rules, allowing property managers to set specific eligibility criteria based on factors such as credit score, criminal history, rental history, and income.

- (1.7) Registered users can open the status of the background check.

3. **Registered User**

- (1.8) Registered users shall be given the option to make posts on the site.

- (1.9) A Registered User shall be able to link their social media accounts such as Instagram and Twitter.

- (1.10) A Registered User shall be able to apply filters when searching for rental listings

- (1.11) A Registered User shall be able to search specifically for number of bedrooms, bathrooms, price, and apartment type.

- (1.12) A Registered User shall be able to use the searchbar for retrieving listings based on a zipcode and address

4. **Location of Rental Listing**

- (1.10) Locations shall include campus area, neighborhood, and proximity to public transport.

- (1.11) Users should be able to filter and view listings by clicking on map markers or drawing custom search areas.

- (1.12) Rental listings shall be separated into regions, where users can view by clicking interactable map region.

5. **Rental Listing:**

- (1.13) Registered Users shall be able to create, update, or delete their rental listings, specifying details such as the number of bedrooms, bathrooms, etc.
- Registered Users shall be able to update/edit their rental listing
- Registered Users shall be able to delete their rental listing
- Registered Users shall be able to specify number of bathrooms
- Registered Users shall be able to specify number of bedrooms
- Registered Users shall be able to specify number of price
- Registered Users shall be able to specify address location
- Registered Users shall be able to specify amenities in details

- (1.14) Registered Users shall be given the permission to add rental listing to the site.

6. **Housing Cost Calculator:**

- (1.15) Registered Users can generate approximate costs for chosen intervals and forecasts for the property.

7. **Posts:**

- (1.16) A post shall be able to contain text
- A post shall be able to contain photos
- A post shall be able to contain links
- A post shall be able to contain user tags
- A post shall be able to contain likes

- (1.17) Posts shall allow comments, each with a unique commentID and creation timestamp.
- A comment shall be allowed to have likes

- (1.18) Posts shall be only edited by the author.

Priority 2:

1. **Registered User:**

- (2.1) Registered users shall save rental listings to their saved properties list.

- (2.2) A Registered User shall be able to form groups of other registered users they wish to house with.

2. **Rental Listing:**

- (2.3) Rental Listing shall have a minimum amount of information and media needed for posting.

3. **Messages:**

- (2.4) Registered users shall be able to initiate private messages with one other registered user.

4. **Saved Properties List:**

- (2.5) Users can create lists of saved properties.

- (2.6) Each saved list shall have a unique ListID.

- (2.7) Properties List shall be under a registered user and not viewable to other non-registered or registered users.

- (2.8) Properties list shall group properties by similarities or by a key word as desired by a registered user.

5. **Groups:**

- (2.9) A group shall share a saved properties list and all receive notifications for a property.

6. **Reports:**

- (2.10) Registered users can generate reports on rental listings, user activity, and feedback.

7. **Housing Cost Calculator:**

- (2.11) The Calculator shall generate a curated list of houses best fit for a registered user based on filters and location.

8. **Reports:**

- (2.12) The system shall provide functionality to generate various types of rental reports, including rental transaction summaries, rental history for specific customers, and rental equipment utilization reports.

- (2.13) Users shall be able to customize the content and format of reports, including selecting specific date ranges, sorting options, and the inclusion/exclusion of specific rental data field, such as rental duration, customer information, and equipment details.

Priority 3:

1. **Registered User:**

- (3.1) Registered users shall send and receive friend requests.

2. **Location of Rental Listing:**

- (3.2) Users may opt to receive notifications when new rental listings by distance from a specified location (e.g., within 5 miles of a particular address or point of interest).

- (3.3) The system shall provide autocomplete suggestions to assist users in specifying the location.

3. **Posts:**

- (3.4) A post shall be able to get pinned by the creator with a maximum of three for his or her account.

- (3.5) A post shall be able to get promoted by the creator at least twice with the boost lasting two days.

4. **Messages:**

- (3.6) The system should allow users to organize their messages, including archiving or deleting old messages.

- (3.7) Users should receive real-time notifications (e.g., push notifications or email notifications) when they receive a new message or when there are updates to existing messages.

5. **Notification:**

- (3.8) Registered users shall receive a notification upon receiving a new message.

- (3.9) A notification shall be able to be turned off or on by a register user or group.

6. **Events:**

- (3.10) Registered users shall be able to set up events (open house, etc) for other users to see.

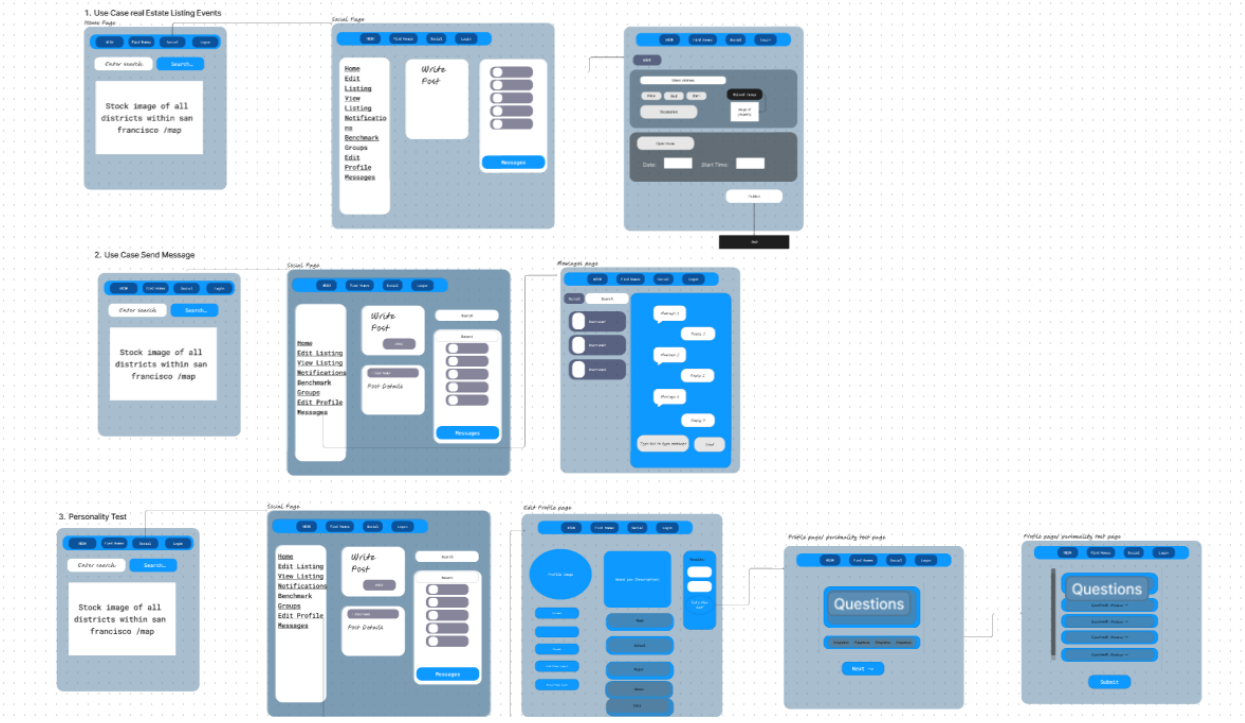
- (3.11) An event shall have its own location, date, eventID, and a list of confirmed attendees.

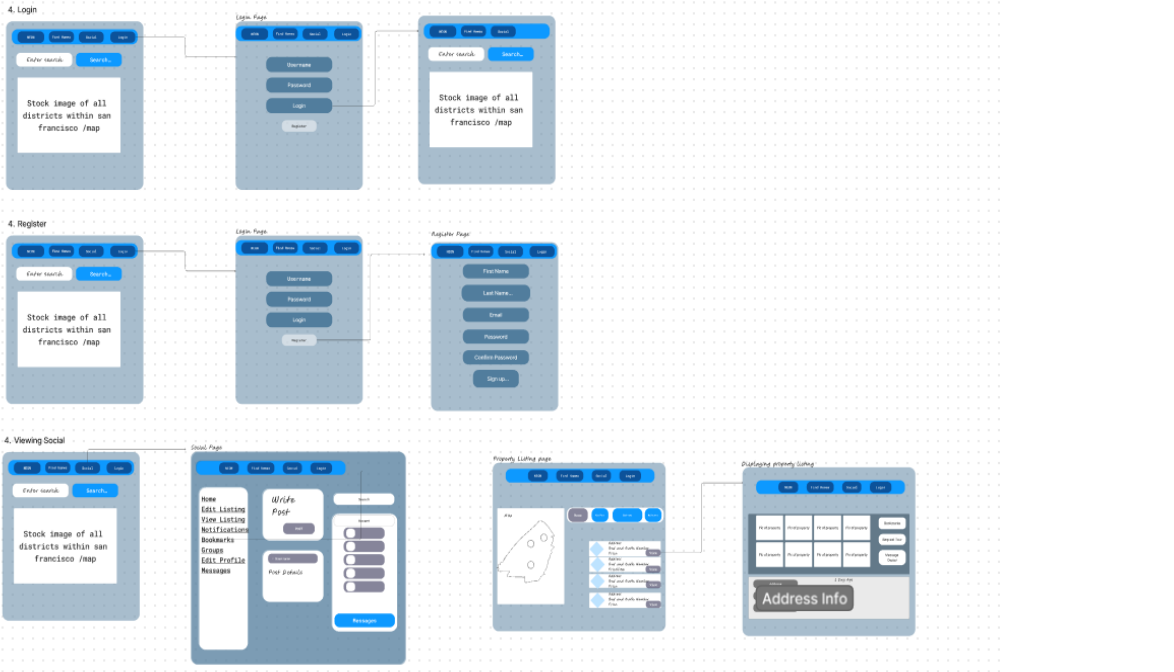
- (3.12) An event shall only be made by verified registered users and have an expected list of attendees.

7. Friend Request:

- (3.13) Register users sending the friend request shall allow permission changes for the requesting friend to view content

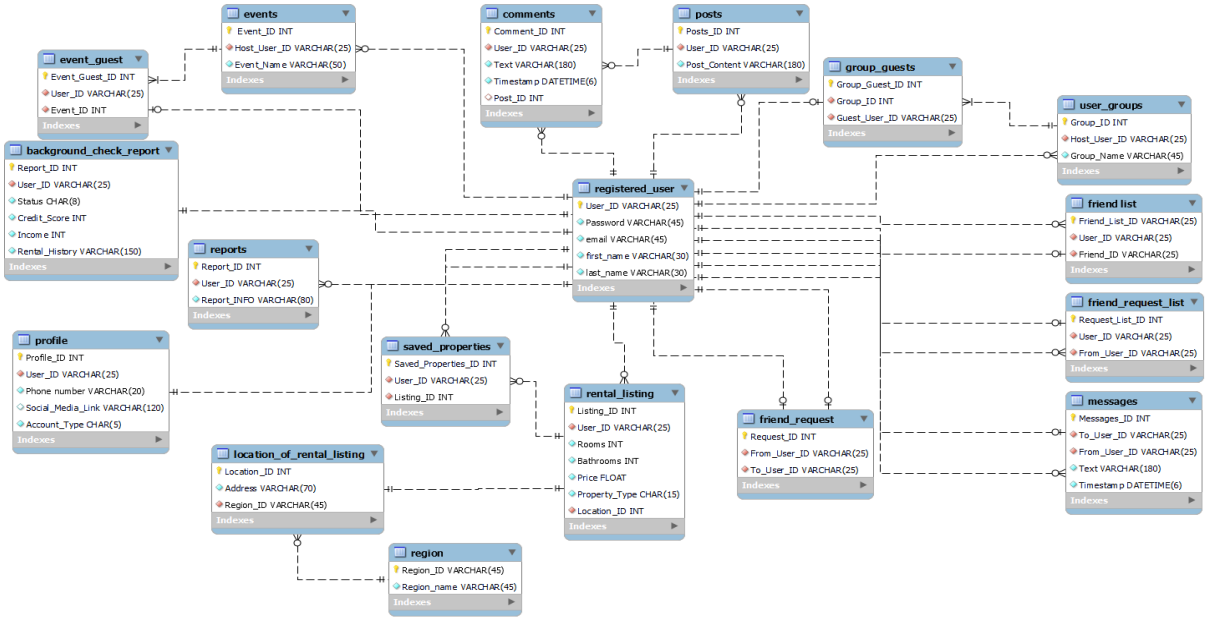
3. Wireframes Based on your Mockups/Storyboards



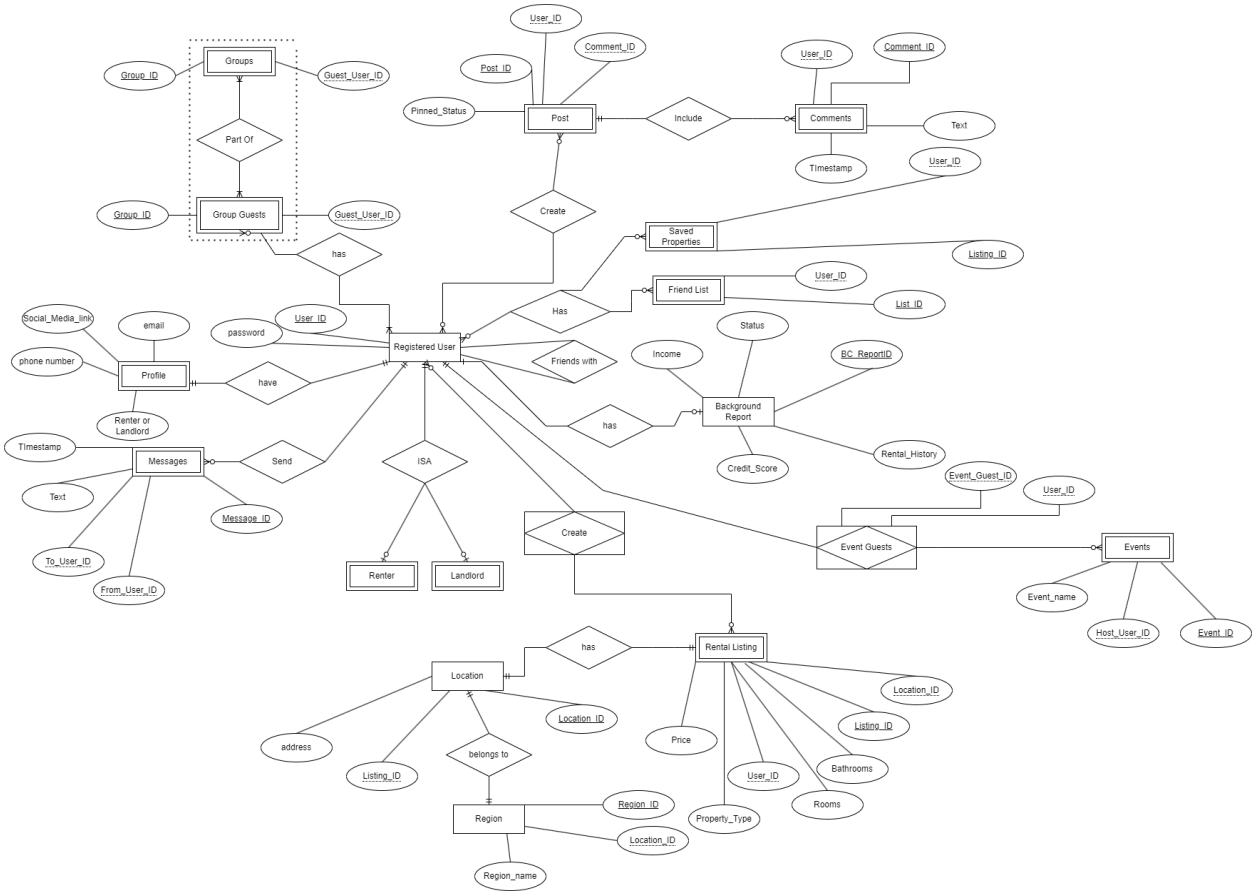


4. High Level Database Architecture and Organization:

EER:

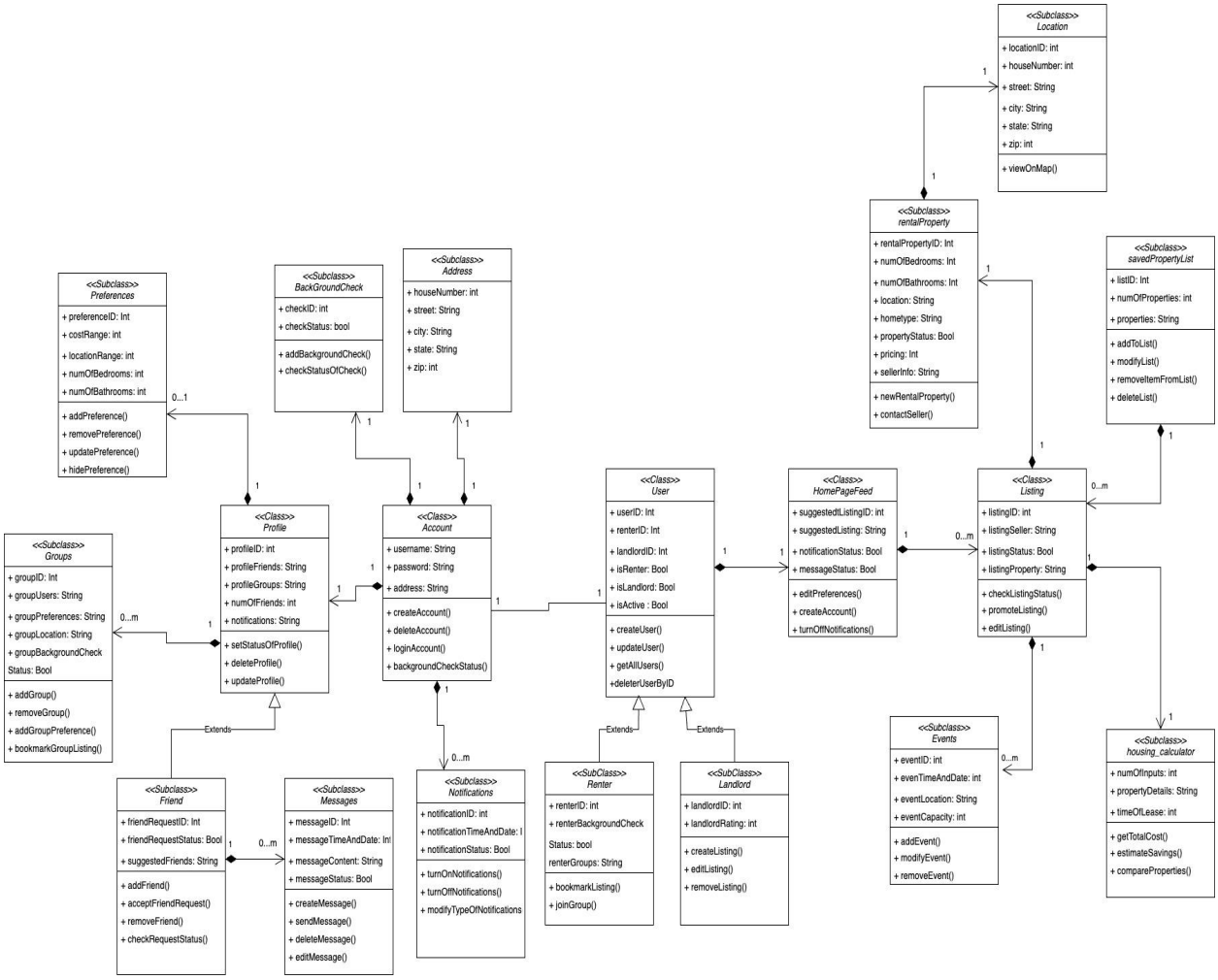


ERD:

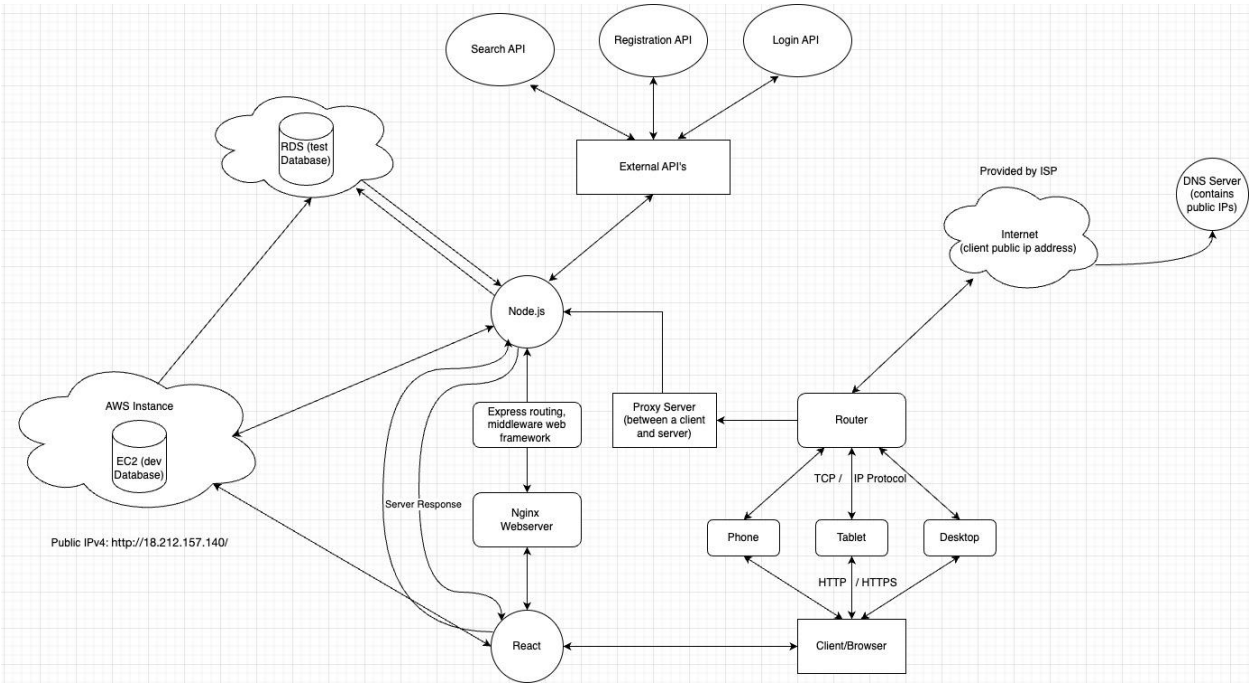


5. High Level Diagrams

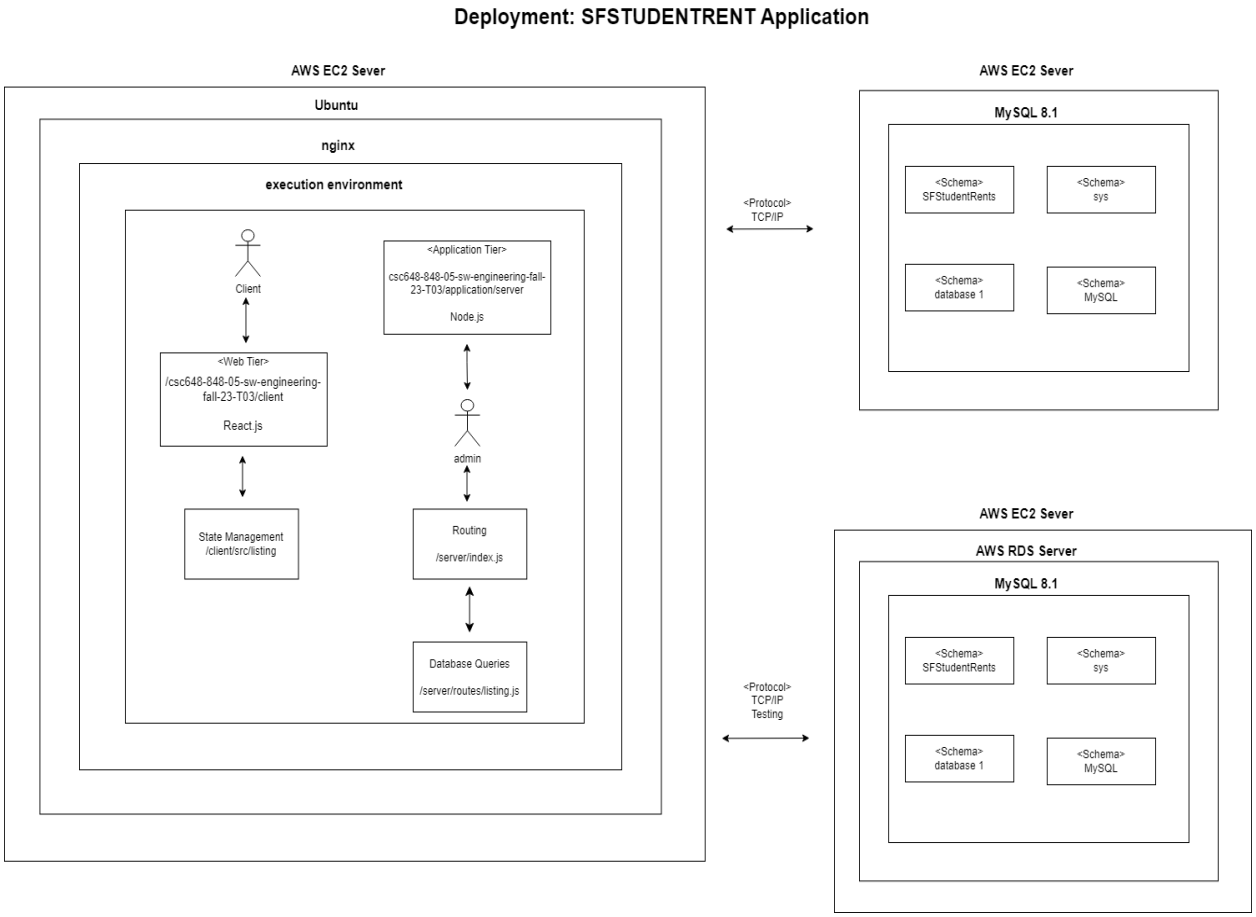
UML Diagram:



Application Network Diagram:



Deployment Diagram:



6. List of Contributions in this milestone

Members	Contributions	Score
Jeremy Tran	● Completed the Registration function for backend and integrated it with the frontend UI component ● Assisted the backend team on connection between backend and database ● Organized and managed all weekly meetings ● Reviewed all work within 1 day according to due dates set by the frontend, backend and database ● Did the M2 revision for storyboard and data definitions	
Geovanni Valadez	● Organized the frontend team and the meetings ● Completed multiple pages and did the UI across the different pages ● Organized the navigation between all pages ● Took on additional responsibilities to help complete the frontend ● Did M2 revisions for section 3	9
Mozhgan Ahsant	● Attended every weekly group meeting and all the subteam meeting ● Attempted the login function	6
Anthony Silva	● Attended every group meeting and was an active participant ● Completed the EER ● Did the database schema and did all revisions needed for the databse	8
Aman Khera	● Organized the backend team and set up project timeline ● Completed the search algorithm, crud functions and login ● Integrated the backend with database and frontend ● Deployed the final product ● Sorted out Git conflicts	10

	● Attended all team meetings and organized backend team meetings	
Ivan Ayala-Brito	● Worked on the frontend pages ● Attempted the storyboard ● Organized the documents for M2V2 and M3V1	7
Daniel Enriqurez	● Set up the Trello for frontend ● Worked on the Figma wirefram	3
Alex Huang	● Worked on and created the crud functions ● Did the search algorithm ● Finished the deployment diagram ● Attending all team and subteam meetings	8

Milestone 4

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

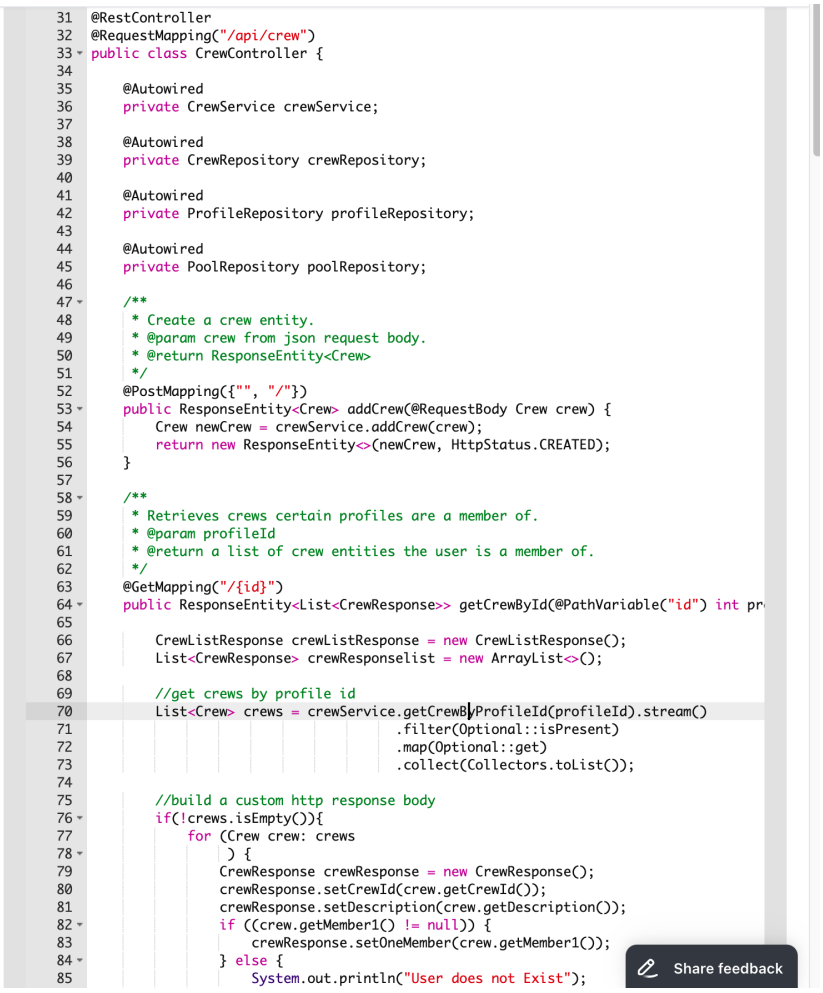
1. Storage capacity for all posts/listings
   1.1 Test to see the database storage for all the posts and listings that are added to the web
2. Hashing Password correctly after login
   2.1 Test to see that after a user enters a password, it is able to hash the password correctly in the database for that 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

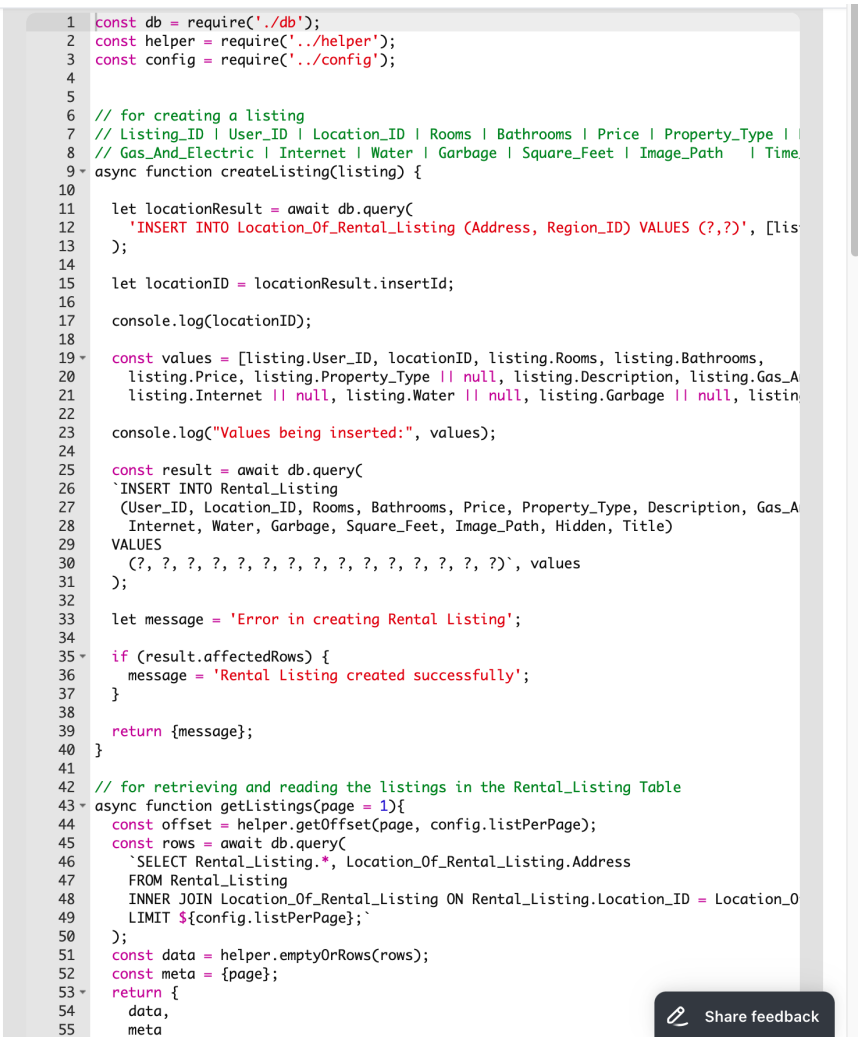
Code Review of 05:

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.

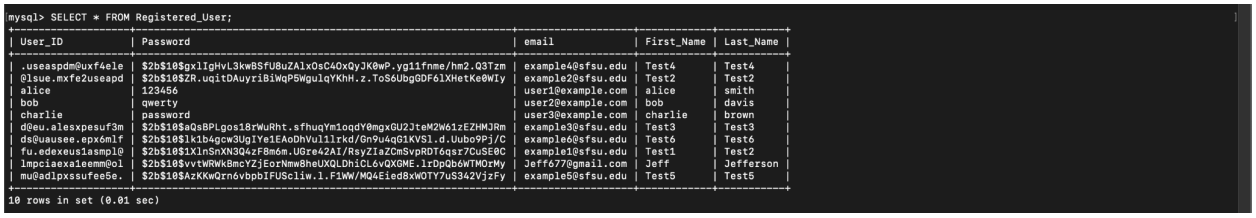
removeListing:

- 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             | First_Name | Last_Name  |
+-------------------+--------------------------------------------------------------+-------------------+------------+------------+
| .useaspdm@uxf4ele | $2b$10$gxlIgHvL3kwBSfU8uZAlxOsC4OxQyJK0wP.yg11fnme/hm2.Q3Tzm | example4@sfsu.edu | Test4      | Test4      |
| @lsue.mxfe2useapd | $2b$10$ZR.uqitDAuyriBiWqP5WgulqYKhH.z.ToS6UbgGDF6lXHetKe0WIy | example2@sfsu.edu | Test2      | Test2      |
| alice             | 123456                                                       | user1@example.com | alice      | smith      |
| bob               | qwerty                                                       | user2@example.com | bob        | davis      |
| charlie           | password                                                     | user3@example.com | charlie    | brown      |
| d0eu.alesxpesuf3m | $2b$10$aQsBPLgos18rWuRht.sfhuqYm1oqdY0mgxGU2JteM2W61zEZHMJRm | example30sfsu.edu | Test3      | Test3      |
| ds@uausee.epx6mlf | $2b$10$lk1b4gcw3UgIYe1EAoDhVul1lrkd/Gn9u4qG1KVSl.d.Uubo9Pj/C | example6@sfsu.edu | Test6      | Test6      |
| fu.edexeus1asmpl@ | $2b$10$1XlnSnXN3Q4zF8m6m.UGre42AI/RsyZIaZCmSvpRDT6qsr7CuSE0C | example1@sfsu.edu | Test1      | Test2      |
| lmpciaexa1eemm@ol | $2b$10$vvtWRWk8mcYZjEorNmw8heUXQLDhiCL6vQXGME.lrDpQb6WTMOrMy | Jeff677@gmail.com | Jeff       | Jefferson  |
| mu@adlpxssufee5e. | $2b$10$AzKKwQrn6vbpbIFUScliw.l.F1WW/MQ4Eied8xWOTY7uS342VjzFy | example5@sfsu.edu | Test5      | Test5      |
+-------------------+--------------------------------------------------------------+-------------------+------------+------------+
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.

Milestone 5:

4. List of Contributions:

Members	Contributions	Score
Jeremy Tran	● Planned out all functional requirements and what needs to be done when ● Planned weekly leads meetings to assess where each team is at and what needs to be completed ● Checked and debugged all the code in frontend and backend ● Did all the connection between frontend and backend ● Created APIs that were missing ● Made changes to frontend pages that weren't dynamic so it could handle backend API ● Helped members of the team with various tasks in private chat	
Geovanni Valadez	● Had great communication with the team and was always responsive ● Attended all team meetings and contributed to the team discussions ● Lead the frontend meetings and planned out the work ● Did all the design in the frontend ● Pages made were dynamic with set variables and routes to be connected to the backend ● Always offered help whether it was in frontend or backend ● Submitted all work on time otherwise notified of any delay	10
Mozhgan Ahsant	● Made APIs for the backend ● Attended all the team and backend meetings	5

Anthony Silva	● Attended every group meeting and was active in the discussions ● Created and made database changes to add new data variables we needed ● Made some changes to the frontend UI component according to Geovanni's task ● Turned in all his work on time unless stated of delay	6
Aman Khera	● Had great communication with the team and was responsive when we needed him ● Attended all the team meetings and contributed to the discussions ● Took on a majority of tasks for the APIs, mostly the harder ones ● Helped teach the members of backend how to run tests ● Made documents to help the backend team with their work ● Planned out specific work in backend and distributed the work	9
Ivan Ayala-Brito	● Created pages in the frontend ● Assisted in the edit of the documentations ● Attended all the team meetings	3
Daniel Enriqurez	● Made backend APIs ● Started attending all the team and backend meetings	4
Alex Huang	● Made backend APIs ● Made changes to frontend UI according to Geovanni's task list ● Attended all the team and backend meetings ● Was active in the team calls ● Organized and commented the code in the backend	7

	Signatures
Geovanni Valadez	Geovanni Valadez
Mozhgan Ahsant	Mozhgan Ahsant
Anthony Silva	Anthony Silva
Aman Khera	Aman Khera
Ivan Ayala-Brito	Ivan Ayala-Brito
Daniel Enriquez	Daniel Enriquez
Alex Huang	Alex Huang

5. Post analysis

There was a lot of things that we learned throughout the project. So to start from the beginning, the first things was that I needed to be a better team lead. I wasn't splitting up the tasks properly and wasn't assigning them to people in proper manner. After that, we managed to still do a pretty good job on the first milestone. In milestone 2, we got a lot better at working on things simultaneously between the prototype and doc. The team was working great starting milestone 2 but things started to fall off when we ran into trouble with the connection between database and backend. The problem was the backend had started working on CRUD operations and connecting too late to debug anything. This was when the backend team learned that they couldn't slack on the assignment regardless of how small the task may seem.

When we got to milestone 3, the backend team was doing better. There was still troubles with connecting to the database but after awhile we were able to find a solution. Everything was going well in milestone 3 once we were able to connect the database, but this is when we started to notice that there were problems outside the technical work. Unfortunately we realized the problem a little too late and that was after milestone 3 submission. The problem with this milestone was that a frontend member went missing and another submitted work 2 days before due date. This didn't give us much time to review or deploy. So when it came time for presentation there was a huge bug that prevented most of our pages to be unreachable.

We realized that a lot of our problems came from people submitting work late and therefore gives us no time to review. On milestone 4, I started doing weekly code reviews and showcase but that didn't work because people still turned in work late or

incomplete. This changed the timeline of our work schedule so showcases weren't always possible before we had to move on. Since we got into actual creations of APIs, a lot of the backend code that was submitted were buggy or didn't hit the use case properly.

In the end, we learned that communication with each other across the teams, frontend, backend and database and among each other is very crucial. Not only would this have made connecting the different components together much easier but also allow other people to plan properly. On top of that, we understand now all our different actions can positively and negatively impact the group as a whole. I, as the team lead learned that I can't always be nice to my team members and I should've been more stern with my group about due dates and expectations. I think if I had better technical knowledge of the full-stack development this could've gone a little bit better.