Design Document

Team 12

Due 2/2/2018

Adam Kogut, Arvindh Swami, Darwin Vaz, Hiten Rathod, Kunal Sinha

Contents

Purpose

New interns often find it a daunting task to find places to stay if they refuse corporate housing. Our team aims to unify and simplify the process of finding a roommate(s) and searching for housing together by using relevant information to allow a user to network with potential roommates and find the perfect place to stay together. Our idea is based on the fact that interns usually have hard preferences on travel time and rent and would like to room with people with similar preferences.

Our functional requirements include the following items:

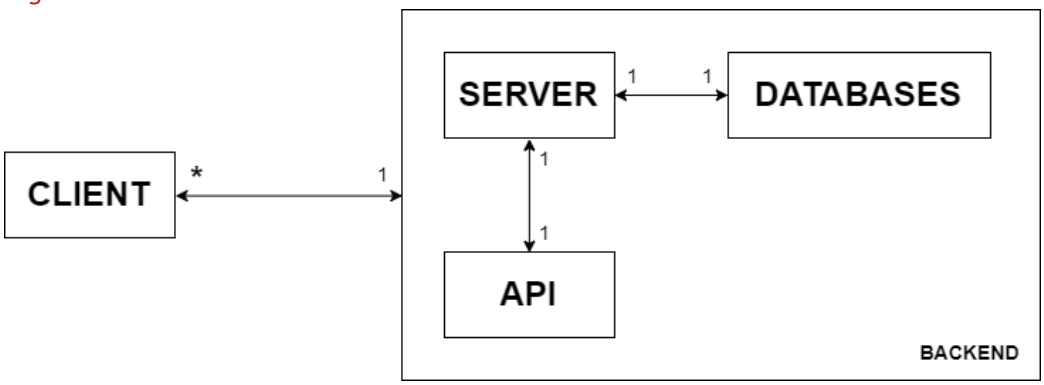
1) Creating a company account
    a) As a company, I would like to be able to create a general company profile
    b) As a company, I would like to be able to have a separate chat for each company location
    c) As a company, I would like to be able to create a moderator account for company moderators
2) Managing the intern class
    a) As a company, I would like to be able to send an email invite link to an intern.
    b) As a company, I would like to be able to keep a master list of invited interns.
    c) As a company, I would like to be able to remove interns from the company when the intern's time at the company is over
    d) As a company, I would like to be able to have all of the company location's interns in a single general chat room.
3) Intern Registration
    a) As an intern, I would like to be able to sign up for Pear using the email invite link.
    b) As an intern, I would like to create my profile with relevant information about myself.
    c) As an intern, I would like to include my preferences for an apartment.
4) Roommate Search
    a) As an intern, I would like to chose the number of roommates I would like to live with.
    b) As an intern, I would like to be able to form a housing group.
5) Housing
    a) As an intern, I would like to only see house listings near the company.
    b) As an intern, I would like to see house listings in my price range.
    c) As an intern, I would like to see distance from the company for each house listing.

d) As an intern, I would like to get notifications about house listings I have selected.
e) As an intern, I would like to see how many people want to book the same place as me.
f) As an intern, I would like to filter houses/apartments according to my criteria.
g) If time allows, I would like to get suggestions for housing based on my housing preferences.
h) As an intern, I would like to have links that directly target rental properties

6) Chatting
   a) As an intern, I would like to be able to join multiple chat rooms based on my preferences.
   b) As an intern, I would like to be able to send private messages to other interns.

Design Outline

High-Level Structure



The client is going to be a slight mix between a thin and fat client. Most of the work is going to be done on the backend. We are still going to cache some data on the local device so that the client doesn't frequently have to communicate with the server thus reducing the number of times the client has to request data from the server.

Depending on what data is being changed by the user, some of it gets cached and then is sent to the server to update the database. For example: if the user is in settings and is updating his preferences, then the updates get cached (the user still sees those changes though), and sent to the server at once. However, if it is something like sending a message, then the 'update' (new message) is sent immediately.

This approach was chosen because after careful considerations. Caching some important things reduces the chances of a DDOS attack on the server. Also, it prevents abuse of the client's (business' and intern's) profile settings as multiple instances of the same logged in person can't all at once change the same thing. It also reduces server-client communication which takes off some burden from the server itself, thus allowing it to serve other clients with more important needs.

Updates sent/received by the client are also going to work on a similar model. We are going to focus on updating the data currently on the user's screen. Even then, we rank them according to time required to importance. For example, seeing a new message is very important for a user, but unless he is currently looking at house listings while chatting, he wouldn't care if the data he is receiving hasn't been updated since the last time he has seen it. This reduces the packet size that the server has to send and thus makes the website appear snappier and more reactive.

The number of details seen by the user is an important factor as well. We want the number to be large (more messages, more house listings updated at the same time) but we also want the updates to be fast enough. So, one idea that we came up with was that we send only a particular amount of details: We say we have a 100 (for the sake of example), but we only show them the top 25 of those. If they want more, as soon as they hit the bottom, we send them another 25, and so on and so forth.

Breakdown of Architecture / Flow of Control

Since we are going to be working with so much confidential information, we can't put enough stress on the security. To prevent any sorts of unauthorized access to the site, all acts (seeing house listings, getting more information about a company) would require you to login. This way, you would already be logged in whenever you come close to confidential information and hence we can easily determine if you have access to it or not.

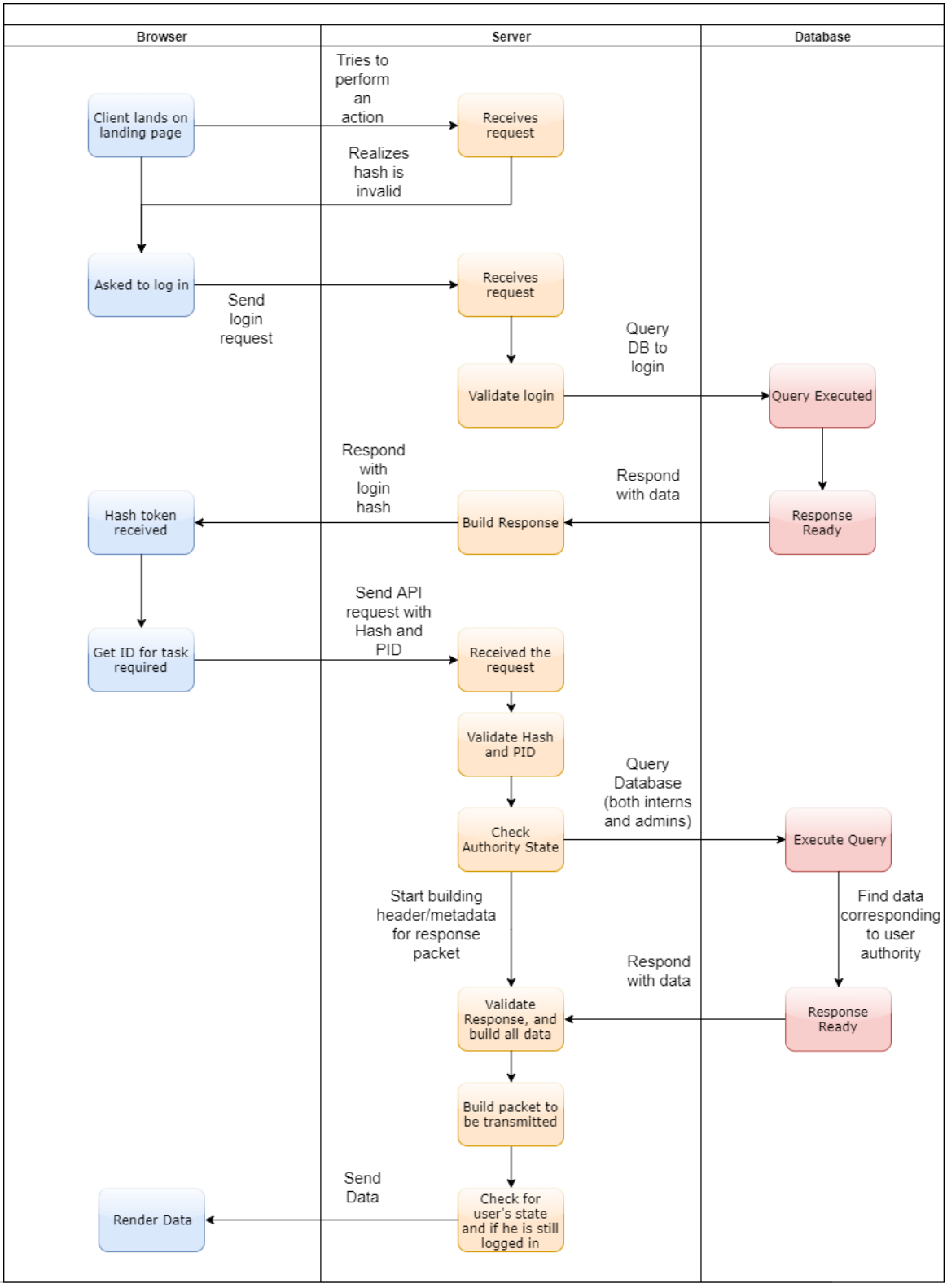
As soon as you login, you will be given a unique hash (the user never sees it, it is just for the server to determine particular access levels and see if they are still logged in) which will act as base for many things on the server side. The Hash will probably be generated such that the number of possible collisions is reduced and thus possible ways to recreate the hash would be almost impossible.

In order to minimize update collisions (two updates to the same piece of data) to the database, that piece of data will be in a temporary 'lock', this will also be used to prevent things like SQL injection.

Whenever the user tries clicking on something, as soon as the server receives it, the server first checks if the user can actually access that piece of data or that part of the website. If they can, then the event is handled, else they get sent an error message. The server queries both databases (the one for interns and the one for admins) at the same time to promote parallelism. While the results return, the server makes the basic metadata/header to respond to the query.

Since we are taking security so seriously, we will try to implement an End-to-End encryption protocol if time allows to make private chats more secure. But since, we will be running the server off of a free site, we would barely have any resources for industry grade encryption-decryption so these features will only be implemented if there is enough time.

A very general flowchart of how the high-level program flow is supposed to look like is given below:


Browser
Server
Database
Tries to perform an action
Client lands on landing page
Receives request
Realizes hash is invalid
Asked to log in
Send login request
Receives request
Query DB to login
Validate login
Query Executed
Respond with login hash
Respond with data
Hash token received
Build Response
Response Ready
Send API request with Hash and PID
Get ID for task required
Received the request
Validate Hash and PID
Query Database (both interns and admins)
Check Authority State
Execute Query
Start building header/metadata for response packet
Find data corresponding to user authority
Respond with data
Validate Response, and build all data
Response Ready
Build packet to be transmitted
Send Data
Render Data
Check for user's state and if he is still logged in

Design Issues

Functional Issues

1) **Should we have a separate login for Interns and Employees?**
   a) Yes
   b) No

   No, we want to keep our website's design simple and not complicate it by having two separate logins for Interns and Employees. Instead, we will keep track of whether the user is an Intern or Employee in the backend when they first create their account.

2) **Should we have a separate landing page for Interns and Employees?**
   a) Yes
   b) No

   Yes, we should because Interns will be a part of multiple chat rooms and have the option to look for housing with potential roommates. However, the Employees will only be part of the company chat and will have more advanced features such as sending referral links to other Interns and deleting messages in the company chat.

3) **Who should be allowed to delete messages?**
   a) Interns
   b) Moderators
   c) Messages cannot be deleted

   We want to give the moderators the option to delete messages if an intern posts an inappropriate message. This feature will only be present in the company chat because the house chat will not have any moderators.

4) **Should banning exist?**
   a) Yes
   b) No

   Yes, but only moderators in their company chats. We want to give the moderators the option to temporarily or permanently ban a user in the off chance that they continuously break the rules of the company chat or they leave the company. As this is only for the moderators in the company chat, they will not be removed from other chats.

5) When and how will users be able to create chat rooms?
   a) Users will automatically be put into chat rooms based on their preferences
   b) Users will be able to create their own private chat rooms with other Users
   c) Users will be able to create their own public chat rooms for anyone to join

   We will put users into predetermined chat rooms based on their preferences and company. They will also be able to create their own chat rooms with people they meet and would like to room with.

Non-Functional Issues

1) What language will we use to program the server?
   a) Node.js
   b) Python
   c) Go

   We will use Go because it is both asynchronous and multi-threaded which will not only make the webapp faster but also make programming easier when multiple users are using the same thing (such as in chat rooms, house lists, etc.).

2) What DBMS should we use to store our data?
   a) MySQL
   b) Firebase
   c) MongoDB

   We will use Firebase, because it is Object-Oriented and stores objects in a hierarchy. Furthermore, it is much easier into incorporate Firebase into the back-end using Javascript. This will make server calls to the database and implementing an API much easier.

3) What front end framework should we use?
   a) AngularJS
   b) React
   c) Ember
   d) Backbone

   We will use React since it is the easiest to use and is the most developer-friendly front-end programming language. It is also great for rendering large amounts of data as it only renders what is changed.

4) How should we store the chat messages in the database?
   a) Linked List
   b) Array
   c) Queue

We will be storing the past 500 messages for any given chat room in the database. Once we reach 500 messages, we will start overwriting the least recent messages. In this implementation, we will have to keep track of the latest message. If we use a doubly linked list, we can move the head to point to the beginning of the chat when messages start to get overwritten. However, an array would use less memory than a doubly linked list. A queue is also great for storing messages. However, we will be able to delete messages much faster using an array.

5) How should users receive notifications?
   a) Email notifications
   b) SMS alerts
   c) WebApp notifications

   When placed in a new chat room, users will receive an email notification. They will receive online notifications (if logged in) whenever any other activity goes on in their account (such as unread messages in chat, new house listing found/added, etc.)

6) How often should chat room messages be updated?
   a) Immediately when a message is sent
   b) 100ms - 500ms
   c) 500ms - 1s

   Chat room messages should be updated every 100ms - 500ms. Messages will be cached and sent to the server in this time frame in order to optimize our server response time. This is crucial since the location based chat rooms will have hundreds of users. This time frame will be negligible for the user and will allow us scale if time allows.

7) What web platform are we going to host our servers?
   a) Heroku
   b) Amazon Web Services
   c) Digital Ocean

Heroku is a free and easy to use web hosting service and it supports Go. AWS is a paid service. Digital Ocean is a good alternative but we are more familiar with Heroku.

8) What type of architecture should we use?
   a) MVC
   b) MVVM
   c) Client-Server Architecture

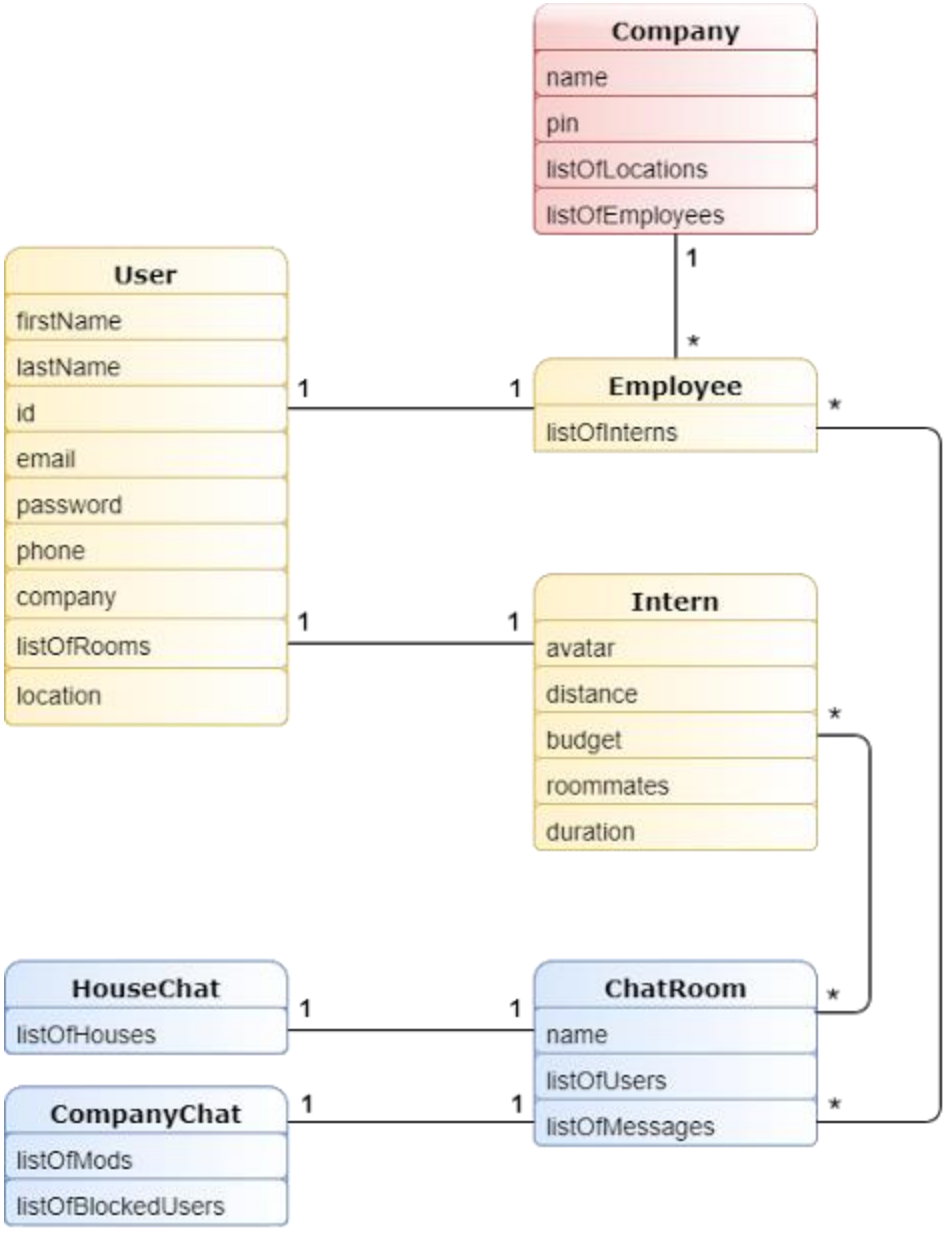
   We will have a Client-server architecture. The webapp will act as the client, while the server will act as the backend.

9) What security will we have?
   a) O-AUTH
   b) SHA-1
   c) RSA

   We will encrypt password and some other important user credentials using SHA-1 encryption algorithm.This will help maintain the integrity of the database and keep user passwords safe. We will store the encrypted passwords in the database. Should the user accounts get hacked the information would be useless to the hacker.

Design Details

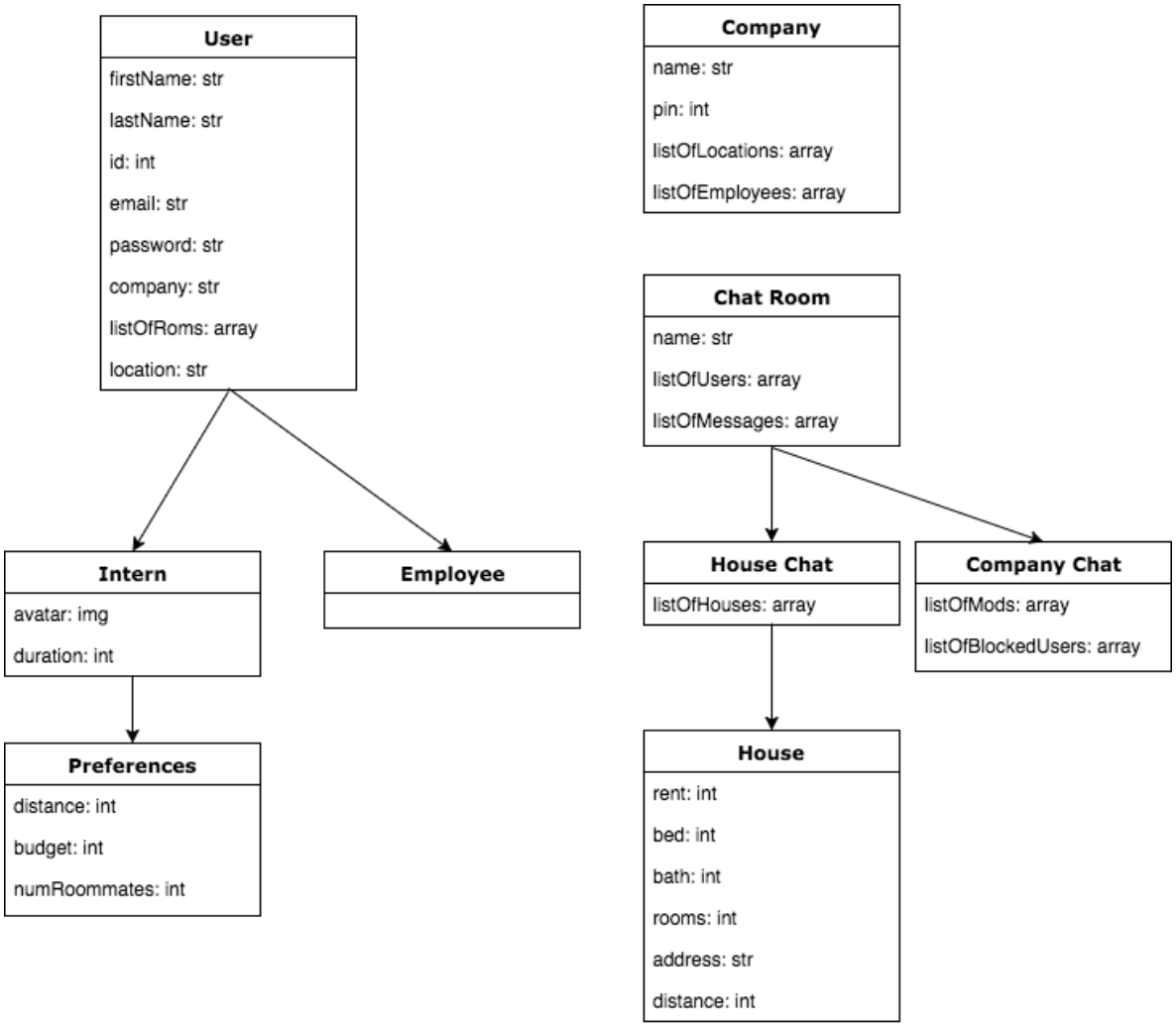
Database (Class) Schema Design



Description of Database Table

- User
  - User is a parent class which contains the user's information including the user's name, email, password, phone, company, location, and the list of chat rooms the user is in.
  - The User class has two children, Intern and Employee, which is specified upon creation of a new account.
- Intern
  - The Intern class inherits all the information from the User parent class.
  - It is created when a referral link is sent to the Intern from the Employee to sign up for finding housing.
  - It includes the Intern's living preferences such as distance from workplace, housing budget, preferred number of roommates, duration of internship and the user's avatar.
- Employee
  - The Employee class inherits all the information from the User parent class.
  - It is created using the registration PIN provided to the Company or by another Employee of the same company.
  - This class also contains the list of interns that the Employee added to our platform.
- Company
  - This class will have one instance per company per location that registers with our platform Pair.
  - It will contain the most important company details including the name, the registration PIN, the list of intern locations, and the list of employees.
- Chat Room
  - Chat Room is a parent class that contains the name of the room, list of chat members, and a limited amount of chat history.
  - There will be multiple chat rooms including one for each Company's location and for the House Chats created for the Interns.
- House Chat
  - The House Chat inherits all the information from the Chat Room parent class.
  - The members of the House Chat are determined based on the preferences set by the Intern on his/her profile page.
  - It also includes the list of houses that were saved (marked as favorites) by the chat members.

- Company Chat
    - The Company Chat inherits all the information from the Chat Room parent class.
    - This chat consists of all the Interns and Employees of a Company for a certain location.
    - This chat also has provisions for the Moderators (which will be the employers). These provisions will allow the Moderators to delete messages and ban other Interns from the chat room.



Security

Due to the confidential information of most of our data, we would need to make sure that almost no data leaks out. Since that would be an ideal case, and this is the real world, no security is perfect. So, in the rare case that data does indeed leak out, we would keep all data encrypted and all data channels to be encrypted. Because of this, data that the server works on is mostly going to be non-sensical to humans.

To prevent things like DDOS attacks, we would cache some information on the client side and transmit only once every ~200 ms (if data is not time sensitive: instead of updating each preference individually, we cache all updates and send it at once) or immediately (~20ms) if it is messages. If time permits I would also like to implement end to end encryption as that would reduce leakage of data through MITM hacking.

Connection to the server

We will be using sockets rather than HTTPS for speed and security. Sockets offer a much faster communication speed which will be really helpful when we make a chatroom as messages need to be sent immediately. This would help us to send and receive messages from the user in a much more secure method and help us shave time off the refresh rate for the site, making the site look snappier.

Server flow of control

The general overview of the server would be that the front end receives a request and passes it onto the backend. The backend sees the request and if the user tries to access any resource without logging in, they would be asked to log in. Once the user has been authenticated and logged in, they would receive a unique hash (a new one generated every time they log in). This would help reduce the times when a person tries to illegally log in and make fake requests to the server.

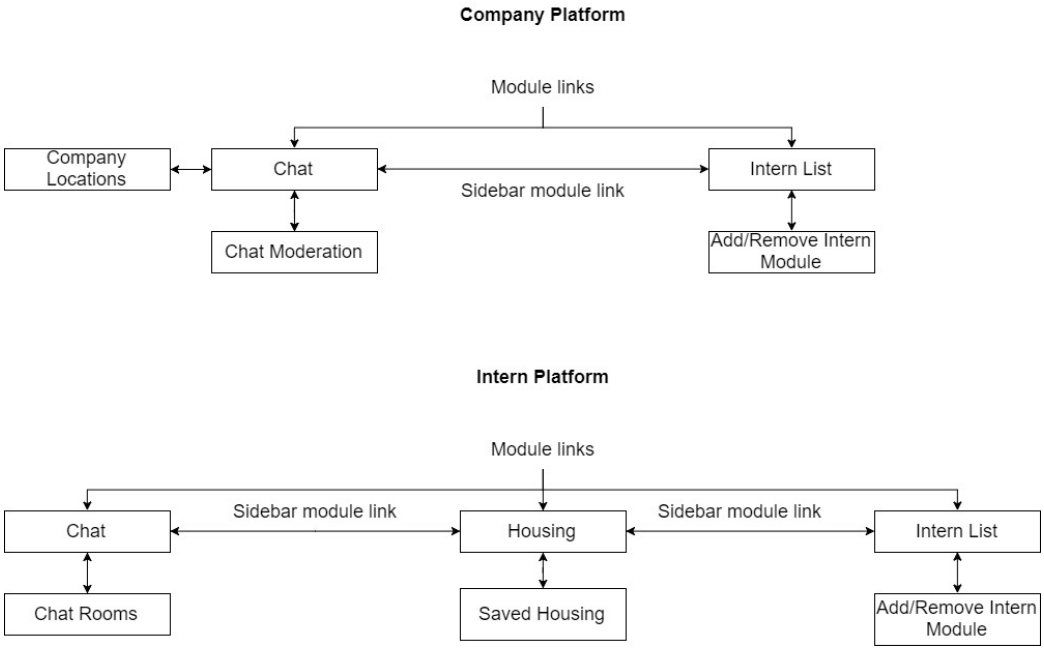
Everytime the user tries to access a resource or clicks on a button (performs an event), the server receives the hash, and verifies it. Once that has been verified with the help of the database, the server tries to query both the databases (one for interns and one for employees) to make sure the user has enough authority to perform whatever the user was trying to do. While the database is trying to get all the data, the server starts preparing the packet to be sent to the front end. This increases parallelism and helps things appear smoother to the user.

User Authentication

When a user registers with us, their login credentials will be stored in our database, and when they login, they will be sent to the server and matched with their details in the database. We will encrypt the passwords before storing them in the database using SHA-1 encryption algorithm. Whenever a user logs in, we will encrypt their password and compare it with the one in the database.

Modules

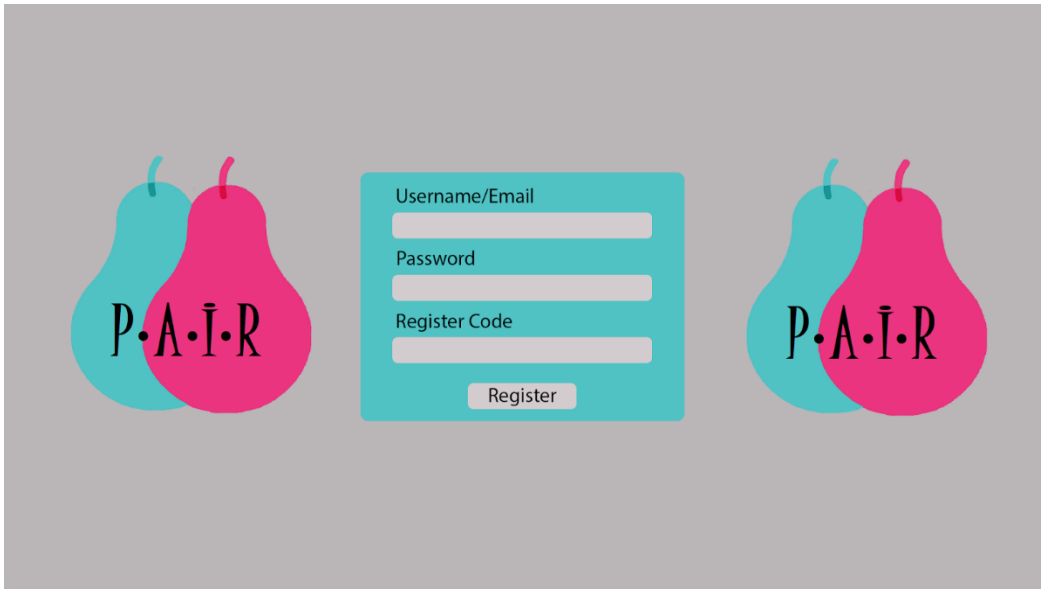
Our navigation emphasizes a simple UI for easy access. The permanent sidebar allows users to access all the modules from any other page of the app. The homepage for our app will the chat module, while the other main modules (Company: Intern List, Interns: Intern List, Saved Houses, Housing Listings) can be accessed by pressing the icons on a singular row of the sidebar. The information displayed will be relevant to the sections selected on the sidebar. For example, when on the chat module, the sidebar links will display the chat rooms a user is connected to, while when on the intern module, it shows the members from the specific room selected, thus acting as a filter.



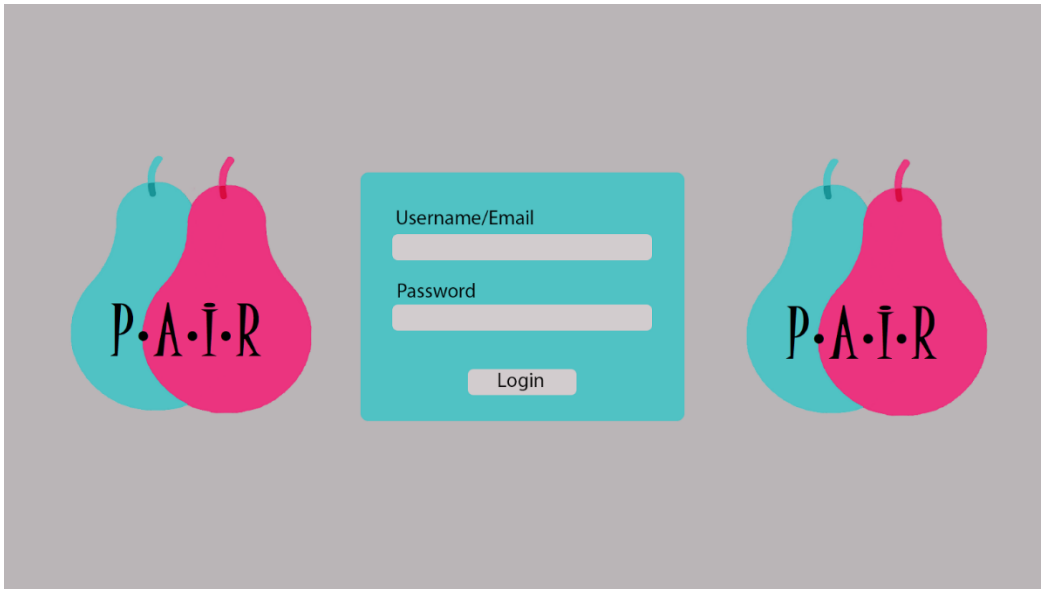
The diagram above demonstrates the way the modules interact using the sidebar links in the company and intern platforms.

UI Mockups

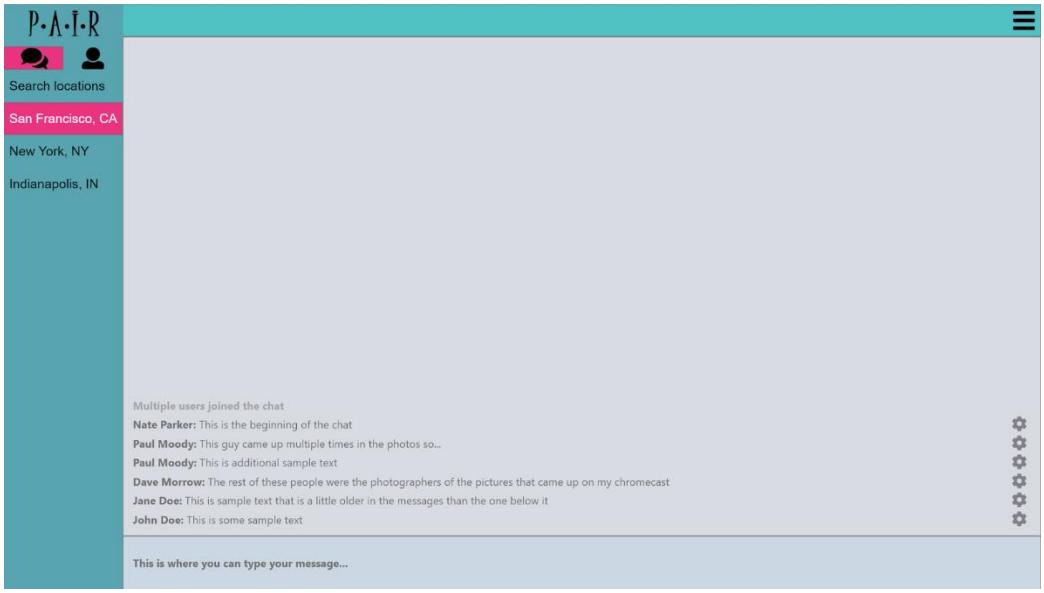
The UI is designed to have consistent elements across the sites. For example, the persistent sidebar allows the users to navigate between the modules at any time with ease. The rest of the UI is built to be neat and instinctive so that the user can navigate through chats, housing lists, and other elements.



Registration screen

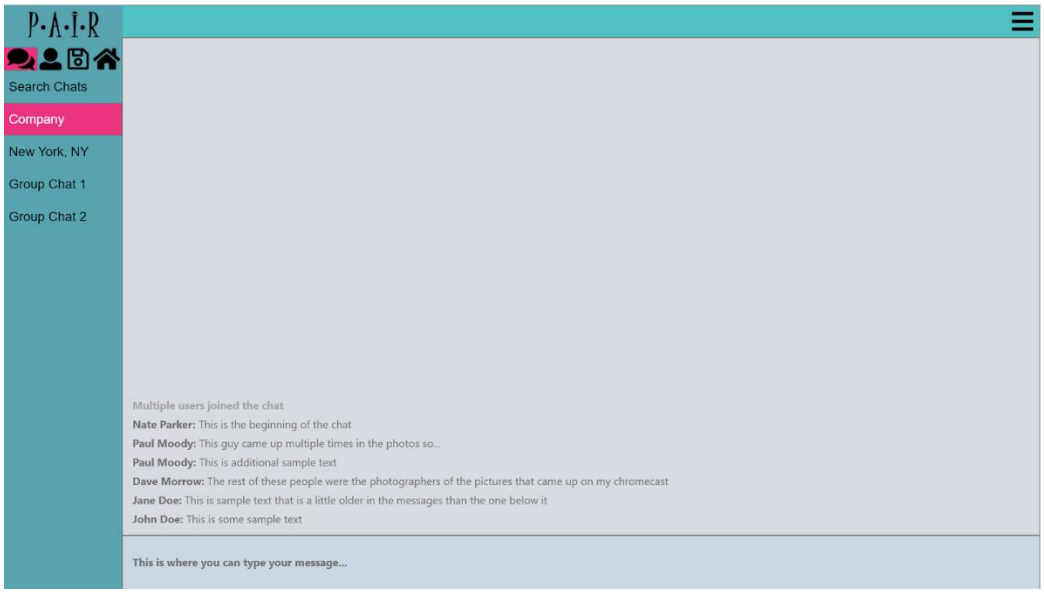


Login



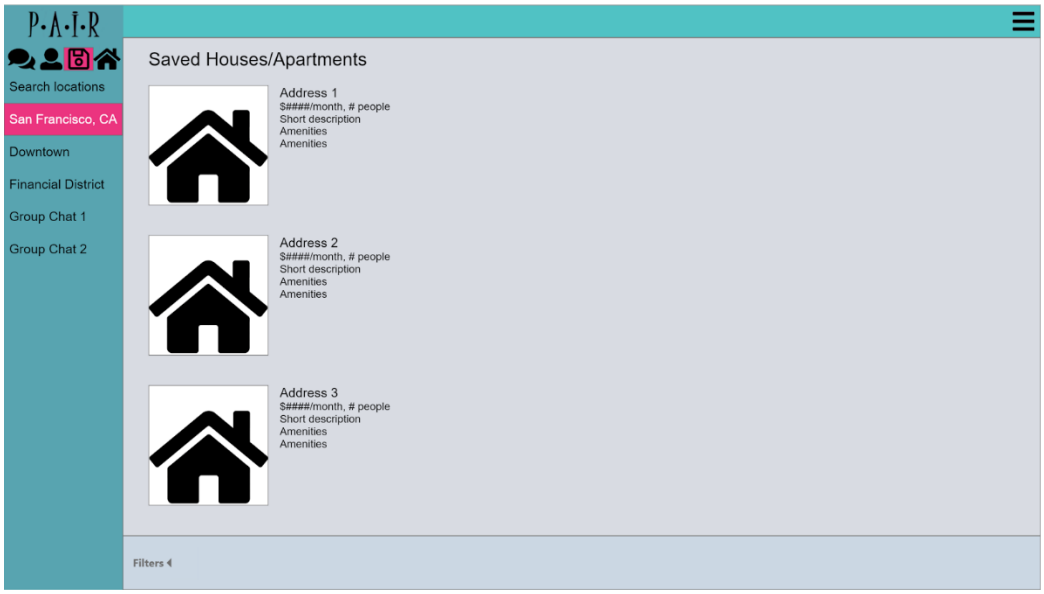
Employee: Chat Room

Employee: Intern list

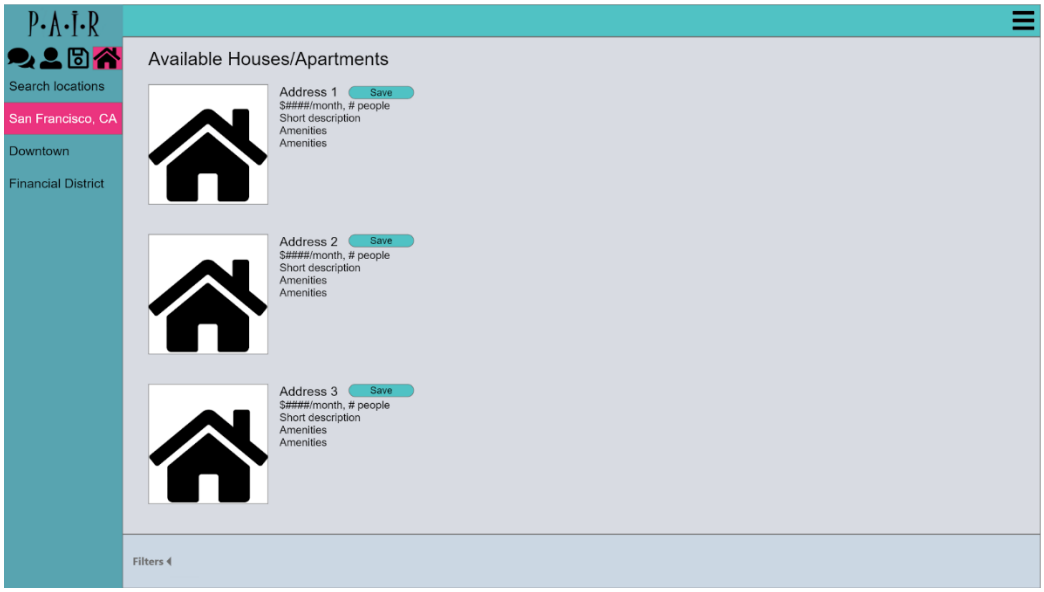


Intern: Chat Rooms

Intern: Intern List



Intern: Saved housing



Intern: Housing Listings