Hesti

Summary

Hesti is an application that connects students who want to sublease their apartments to students who need subleases. Users can use Google authentication to sign up and they will be placed under the correct organization that their email is associated with. This ensures that users are communicating with other students in the same community. Users who have a place to sublease can post a listing about it. Users who want a sublease can scroll through the listings and can either favorite or request them. When they favorite a listing, it's essentially just bookmarking the listing to look at later. When they request a listing, the user who listed that listing will be able to see their contact information so that they can further discuss/negotiate a possible deal.

The main customer need was an application to provide an easy and reliable way for students to find subleases and subletters. We do this by allowing users to search and filter through the available listings within their organization. Students can communicate with each other while keeping their identities and addresses private for safety until they are ready to make an agreement. The application would need to be tailored towards students, and only allow students to post or request listings. This is so that they do not need to worry about larger companies attempting to advertise their apartments. We meet this need by using google authentication to ensure that each user is an individual with a student email and grouping them together by their organization/community. Additionally, we created an admin view that manages and has elevated permissions for the application such as changing the color scheme of the application to match their organization's branding. The stakeholders of this application are students looking to sublease an apartment, students looking to sublet an apartment, and university administrators who want to provide a safe service for their students.

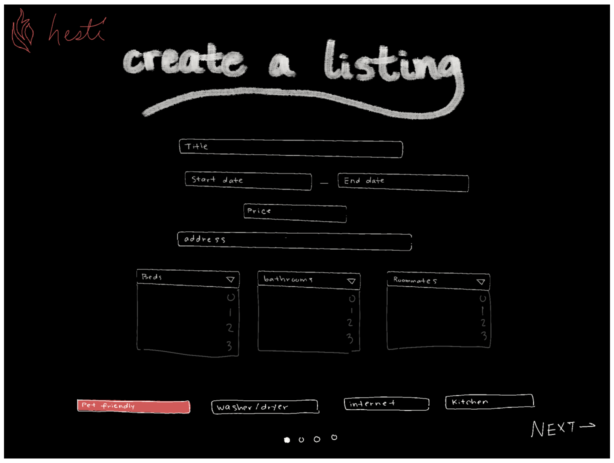
User Stories with Lo-Fi UI Mockups/Storyboards

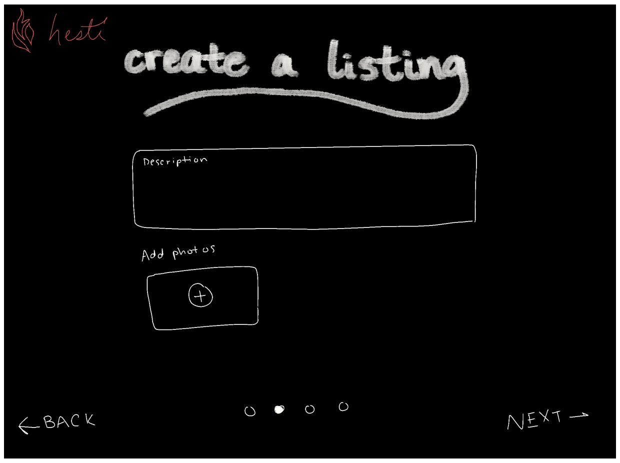
User Stories and Associated Mockups

Create an account. The user should be able to create an account so that they can search through and post listings. This feature was given 4 points because it is a core part of our service and is required for the majority of the interactions. This story has been implemented. This story originally included logging in, which was moved to its own story because it was one of our first large features and would likely involve a lot of setup.

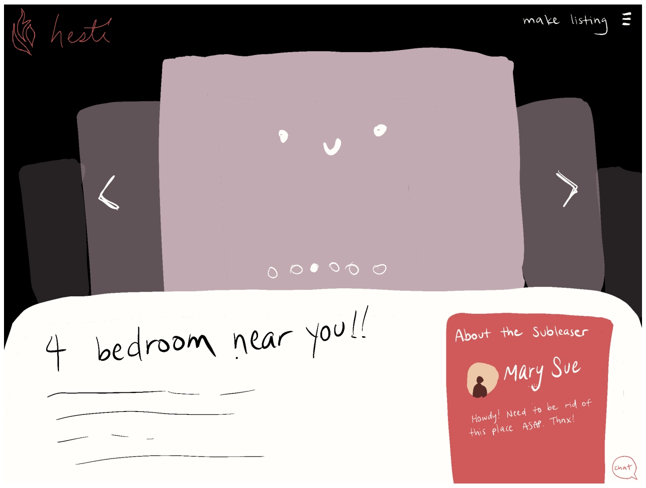
Log in. The user should be able to log into a preexisting account. This feature was given 1 point because it was as simple as making sure the username and password that were filled in had a matching entry in our database, then loading the correct session. Most of the setup needed for this story was done for the "create an account" story. This story has been implemented. We later added Google authentication to our login system, so that users had the option to log in with their school's Gmail account instead of entering their email and password manually.

Create a listing. The user should be able to create a listing by filling out property details so that they can find someone to sublease the property. This story was given 2 points because it is also a core part of our service. However, it was relatively simple, as it is processing a form and creating an entry in our database. This story has been implemented. There were little to no changes to this story.

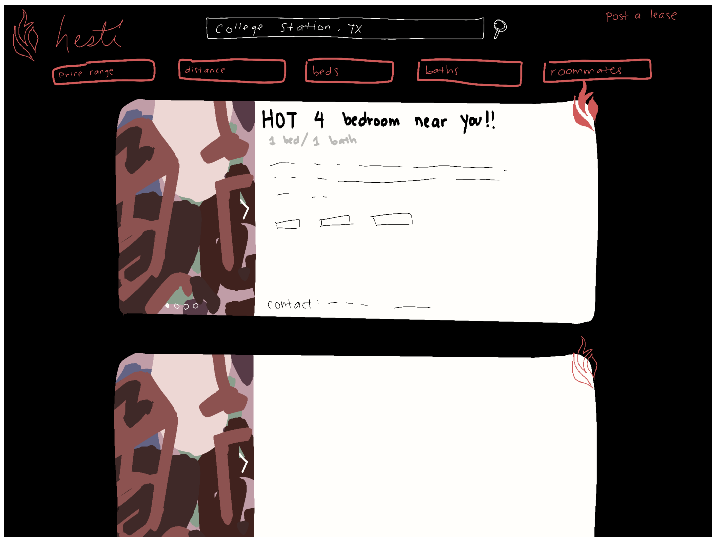




View a listing. The user should be able to view all information of an individual listing (e.g. amenities, location) so that they can determine whether they want to request to sublease the listing. This story was given 3 points because it involved finding and pulling the data from our database and presenting it to the user. This story has been implemented. There were little to no changes to this story.



View all listings. The user should be able to scroll through all listings and view a summary of details for each listing. This story was given 3 points because it involved finding and pulling the data from our database and presenting it to the user. This story has been implemented. There were little to no changes to this story.



Amenities. The user should be able to view the amenities of an apartment or property in the listing. This story was given 3 points because we had to come up with a good way to store these amenities in our database, which ended up being a separate table from our regular listings table. This story has been implemented. There were little to no changes to this story.

View account information. The user should be able to view the information associated with their account (e.g. name, email, username). This story was given 2 points because it required pulling data from our database based on the existing session and displaying it to the user. This story has been implemented. There were little to no changes to this story.

Edit account information. The user should be able to edit and update the information associated with their account. This story was given 3 points because it was similar to the "view account information" story, except it required updating the data rather than simply displaying it. This story has been implemented. There were little to no changes to this story.

Delete account. The user should be able to delete their account when they want to stop using this service. This story was given 2 points because it was similar to the "view account information" story, except it required deleting the data from our database. This story has been implemented. There were little to no changes to this story.

View my listings. The user should be able to view and manage the listings they have created. This story was given 2 points because it was similar to the "view all listings" story, except it was filtered by listings that the specific user had created. This story has been implemented. There were little to no changes to this story.

Edit listing. The user should be able to update the information in their listings.
This story was given 3 points because it was similar to the "edit account information" story. This story has been implemented. There were little to no changes to this story.

Favorite a listing. The user should be able to "favorite" listings, which are saved and can be reviewed at a later time. This story was given 3 points because we had to create a system that kept track of listings that were favorited for each account, have it persist, and be retrieved later. This story has been implemented. There were little to no changes to this story.

Delete listing. The user should be able to delete a listing so it can no longer be viewed by others. This story was given 1 point because it involved finding the correct entry in our database associated with the listing and removing it from the database. This story has been implemented. There were little to no changes to this story.

View favorited listings. The user should be able to view the listings they favorited previously on one page. This story was given 1 point because most of the initial setup for this story was done in the "favorite a listing" story. The only work left was to actually display it back to the user. This story has been implemented. There were little to no changes to this story.

Password confirmation on signup. The user should be required to confirm their password when signing up for the service. This story was given 1 point because this was as simple as adding an extra "confirm password" field to the sign up page and making sure the entries matched. This story has been implemented. There were little to no changes to this story.

User-friendly form validation on signup. The user should be made aware of any errors they made when creating an account (e.g. username already exists). This story was given 2 points because the validation was already implemented in the backend, but now had to be displayed to the user. This story has been implemented. There were little to no changes to this story.

Filter available subleases based on preferences. The user should be able to apply a filter to the available listings to make searching easier. No points added to this story because it would have too large points, so it was split into multiple smaller stories. This story was implemented under separate stories.

Filter available subleases based on private or shared bedroom. The user should be able to apply a filter to the available listings for whether the bedroom is shared or private. This story was given 2 points because this feature required pulling only the listings that have either private or shared bedrooms. This information was already collected and stored in our database. This story has been implemented. There were little to no changes to this story.

Filter available subleases based on private or shared bathroom. The user should be able to apply a filter to the available listings for whether the bathroom is shared or private. This story was given 2 points because it is similar to the "filter available subleases based on private or shared bedroom" story. This story has been implemented. There were little to no changes to this story.

Filter available subleases based on amenities. The user should be able to apply a filter to the available listings based on the amenities they want. This story was given 3 points because it is similar to the "filter available subleases based on private or shared bedroom" story. One difference is that it required filtering through multiple features instead of just one. This story has been implemented. There were little to no changes to this story.

View number of users who favorited a listing. The user should be able to view how many other users favorited their listing to gauge interest. This story was given 2 points because we had to add this field to our listing information in our database and keep it updated. This story has been implemented. There were little to no changes to this story.

Unfavorite a listing. The user should be able to unfavorite a listing they had previously favorited if they decide they are no longer interested. This story was given 2 points because it is relatively simple to undo the changes made when favoriting a listing. This story has been implemented. There were little to no changes to this story.

Create admin account. An organization should be able to create an admin account so that they can monitor the activity for their organization. This story was given 1 point. To officially create an admin account, the only requirement was to set their admin field in the database to true. The specific features an admin account had access to were given their own stories. This story has been implemented. There were little to no changes to this story.

Edit organization settings. The admin should be able to edit the color scheme of their organization's site so that they have a unique aesthetic from other organizations. The story was given 2 points because it required separating between organizations and making sure that the settings were applied to all users within the same organization. This story has been implemented. There were little to no changes to this story.

View listings limited to organization. The user should only be able to view listings from other users within the same organization so that they only see listings that are relevant to them. The story was given 2 points because this also required separating users based on organization and only pulling data for listings that apply to the user. This story has been implemented. There were little to no changes to this story.

Search available subleases. The user should be able to search through the available subleases using a search bar. This story was given 3 points because we had to figure out which fields the search bar would actually sort through (e.g. title, description) and make sure it was implemented in a way that would make sense to the user. This story has been implemented. There were little to no changes to this story.

Request a listing. The user should be able to request a listing for a sublease. This story was given 3 points because we had to figure out what is meant by requesting a sublease and what private information should be made available when requesting a sublease while also maintaining safety. This story has been implemented. There were little to no changes to this story.

Cancel a request. The user should be able to cancel a request for a sublease in case they change their mind. This story was given 2 points because it is relatively simple to undo the changes made in the "request a listing" story. This story has been implemented. There were little to no changes to this story.

View pending requests. The user should be able to view any requests for their listing so that they can contact each other and come to an agreement. This story was given 2 points because the data for requests was already collected, and only had to be displayed to the user. This story has been implemented. There were little to no changes to this story.

Filter by price range. The user should be able to filter the available listings based on an entered price range. The story was given 2 points because this was similar to previous stories about filtering listings. This story has been implemented. There were little to no changes to this story.

Add pictures to listing. The user should be able to add pictures to their listing or view pictures attached to an existing listing. This story was given 4 points because we struggled with coming up with a way to store and retrieve the photos as well as making sure the pictures persisted. This story has been implemented. There were little to no changes to this story.

SSO login. The user should be given the option to sign in with a Google account so that they can be placed in an organization. This story was given 4 points because it involved learning how to work with SSO, as well as separating users out into organizations. Google does not provide a built-in way of doing this, other than a single organization (TAMU in this case) or no filtering at all. We had to implement this filtering logic in our app. This story has been implemented. There were little to no changes to this story.

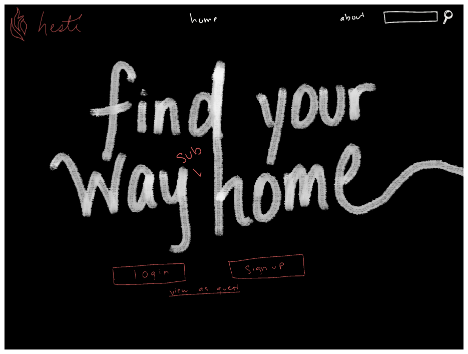
Photo gallery. The student should be able to view photos attached to a listing. No points added because this story was merged with the "add pictures to listing" story. This story is implemented in a different story.

Manage listings. The admin of an organization should be able to manage all listings for their organization as a form of moderation. There were no points added to this feature because we ran out of time to add points and implement it. This story was removed from our final scope of the project.

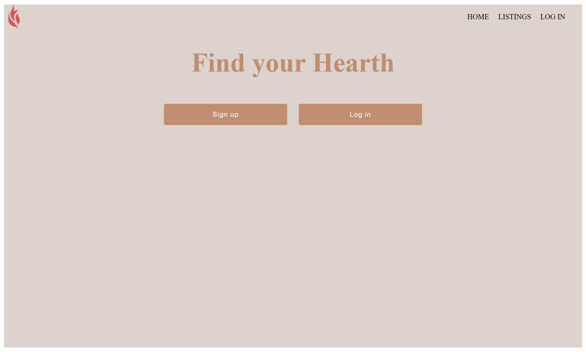
Manage user accounts. The admin of an organization should be able to manage other accounts within their organization as a form of moderation. There were no points added to this feature because we ran out of time to add points and implement it. This story was removed from our final scope of the project.

Sort available listings. The user should be able to sort available listings by parameters such as price, date added, etc. There were no points added to this feature because we ran out of time to add points and implement it. This story was removed from our final scope of the project.

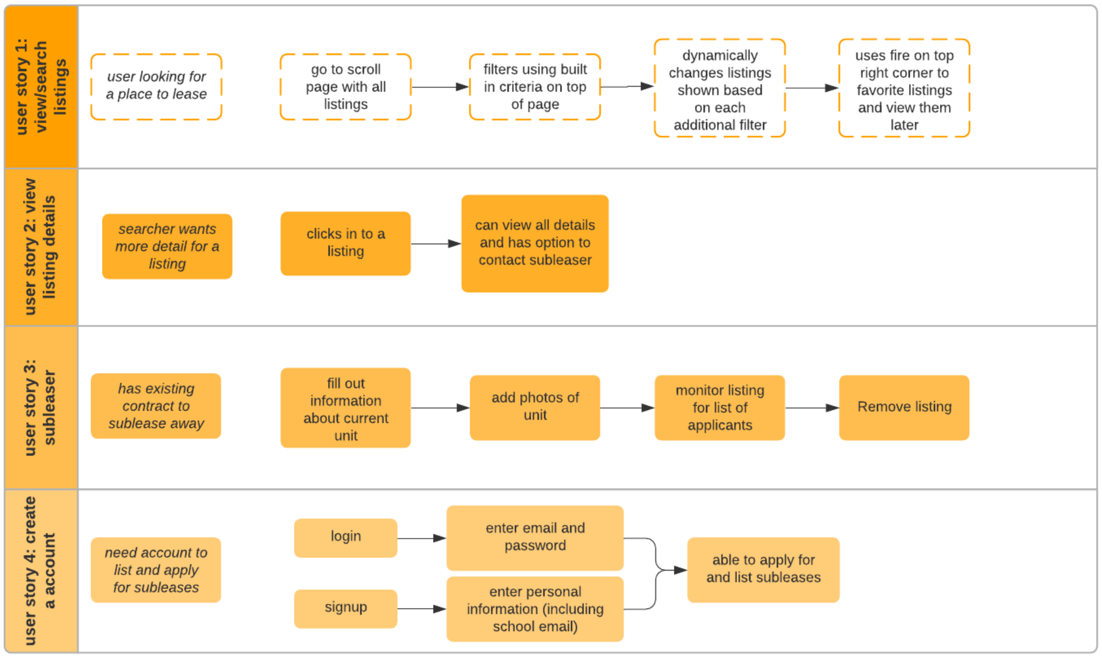
Initial Mock Up for Home Page



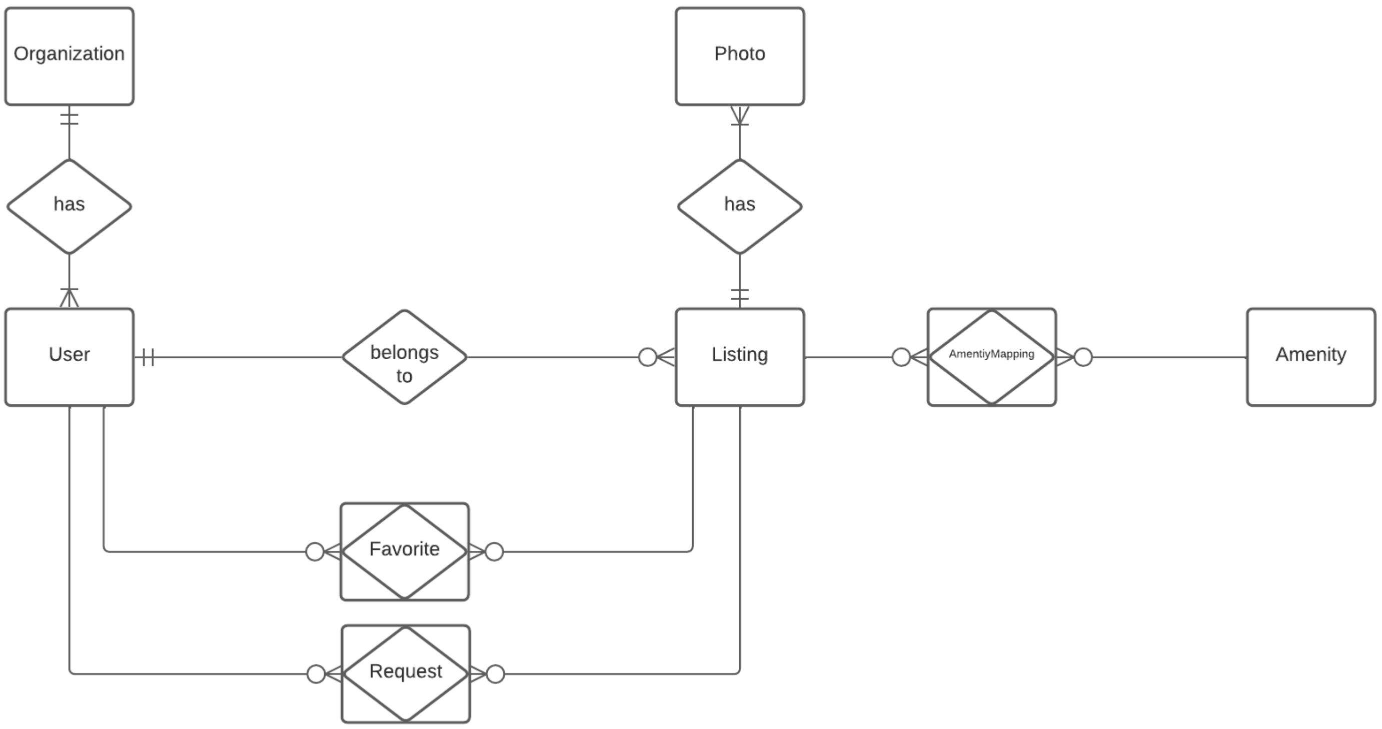
Final Home Page



Storyboard


user story 1: view/search listings
user looking for a place to lease
go to scroll page with all listings
filters using built in criteria on top of page
dynamically changes listings shown based on each additional filter
uses fire on top right corner to favorite listings and view them later
user story 2: view listing details
searcher wants more detail for a listing
clicks in to a listing
can view all details and has option to contact subleaser
user story 3: subleaser
has existing contract to sublease away
fill out information about current unit
add photos of unit
monitor listing for list of applicants
Remove listing
user story 4: create a account
need account to list and apply for subleases
login
signup
enter email and password
enter personal information (including school email)
able to apply for and list subleases

Database Diagram



Team Roles

Scrum Master ➔ Carolyn Nguyen
Product Owner ➔ Mariel Lopez
These roles persisted throughout the whole project.

Scrum Iterations and Customer Meetings

Iteration 0

- Finalized client requirements and restraints
- Created lo-fi mockups for the application to ensure usability and design flow between pages
- Made list of user stories to be implemented through each iteration
- Designated developer roles between team members
- Defined Tech stack
- Performed initial setup and API connections
- Corresponding customer meeting: **December 30, 2021 at 10:30 am, YMCA Building**
    - The client expressed the idea and purpose of the application that he wanted and what features he wanted to see. There were no stories or software to demo for this meeting.

Iteration 1

- Created an initial design diagram for the outline of the application
- Outlined a database design through a UML diagram to provide a visualized view of the system
- Completed 4 user stories:
    - Create an account (4 points)
    - Login (1 point)
    - View a listing (3 points)
    - Create a listing (2 points)
- Corresponding customer meeting: **January 5, 2022 at 7:00pm, The House of Grandma**
    - The team demoed all completed user stories during the iteration. At this point, the user could only add title, address, rent, lease term, floor plan, bedroom and bathroom privacy, and description. The client expressed his wishes to add more branding to the application.

Iteration 2

- Updated the UML diagram to reflect actual database
- Completed 11 user stories:
    - Password confirmation on signup (1 point)
    - User-friendly form validation on sign up (2 point)
    - View account information (2 points)
    - Edit account information (3 points)
    - Delete account (2 points)
    - View all listings made by an individual user (2 points)
    - Edit a listing (3 points)
    - Delete a listing (1 point)
    - Favorite a listing (3 points)
    - View favorited listings (1 point)
    - Search listings based on amenities available (3 points)
- Corresponding customer meeting: **January 21, 2022 at 6:00pm, PETR 414**

    - The team added the app logo to each page to improve branding. The team also demoed all completed user stories during the iteration. Password confirmation and form validation user stories were added to the backlog.

Iteration 3

- Updated the UML diagram to include photo database table
- Updated the design diagram to account for refactoring
- Completed 5 user stories:
    - Filter available subleases based on private or shared bedroom (2 points)
    - Filter available subleases based on private or shared bathroom (2 points)
    - Filter available subleases based on amenities (3 points).
    - View Number of Users who Favorited a Listing (2 points)
    - Unfavorite a Listing (2 points)
- Corresponding customer meeting : **February 7, 2022 at 12:45pm, PETR 414**
    - The team demoed all completed user stories during the iteration. We were unable to complete stories that allowed users to search through available subleases, SSO login, and adding pictures to a listing. The client expressed his interest to see 3 or 4 accounts to see the interaction between accounts.

Iteration 4

- Updated the UML to include admin functionality
- Updated the design diagram to establish the organization view
- Photos:
    - When a user is creating a new listing, the last step is to browse their library for a photo. The user will click the “choose file” button and a popup of their file will appear. After selecting a photo to attach, the popup closes and the user may create the listing. The user is then redirected to the listing’s individual where their photo is displayed. When the user goes back to the main page of all listings, the photo is the thumbnail of the listing. If one of these things are selected, the user is again redirected to the individual listing view and the image can be seen in full.
- SSO login authentication
    - Single Sign On with Google was partially implemented. Credentials have been created and added to the web app. The current options we have seen so far are “Internal” (TAMU only) or “External” (anyone).
- Completed 4 user stories:
    - Create admin account (1 point)
    - Edit organization settings (2 points)
    - View listings limited to organization (2 points)
    - Search available subleases (3 points)
- Corresponding customer meeting: **February 23, 2022 at 1:00pm, Piada**
    - The team demoed all completed stories during the iteration. We were still unable to demo adding photos to listings and SSO login. We demoed with a larger test data set to display the interaction between users. The client mentioned that he wanted to see photos stored in Amazon S3, filtering by price range, SSO login, and requested features that happen between users who have listings and users who want to sublease those listings.

Iteration 5

- Updated the UML to include leasing contract requests
- Updated the design diagram to connect a user to a listing through a sublease request
- Completed 6 user stories:
    - Request a listing (3 points)
    - Cancel a request (2 points)
    - View pending requesters (2 points)
    - Filter by price range (2 points)
    - Add pictures to listing (4 points)
    - SSO login (4 points)
- Corresponding customer meeting: **March 10, 2022 at 1:00pm, Piada**
    - The team demoed all completed stories during the iteration.

BDD/TDD process

We used a mix of BDD and TDD for this project. We would first create basic tests for every user story. After completing a user story, we would create more tests that apply to the story. We often had to add more tests after completing a user story because a lot of times a user story would overlap with other user actions. Behavior driven development really helped us understand whether or not we were meeting the customer's needs. It helps clarify what the customer really wants to see. It also helped show unexpected behavior for certain actions. Some problems that arose from BDD was that sometimes it was difficult to tell why a certain test was failing because there could have been many functions involved in the test. Test driven development helped us debug whether specific functions were working the way we wanted them to. It had less of a connection to the client's vision and tested if our code actually worked.

Configuration Management Approach

Spikes

We had a spike to investigate how to set up our database systems. We were all new to working with databases so we didn't know which one was the best option. For this spike, we investigated using Amazon RDS, Heroku PostgreSQL, and SQLite3. We deemed that SQLite3 would not suffice for production and long term purposes since it was just a local file, but it could possibly be used for testing. We ended up using Amazon RDS for both our dev and test databases and Heroku PostgreSQL for our production since we would be deploying to Heroku anyway.

Branches and Releases

In total, we created **31 branches**. There are a total of **6 releases**, 1 for every iteration including Iteration 0 which consists of the setup. For each iteration, we created a new branch for each user story group. For example, setting up the user crud operations consisted of four different user stories, but they all were done in the same branch. Branches were also created to write tests and fix our user interface.

Tools Used

Github
We used Github for our source control. Using Github went smoothly for the most part, save for the occasional merge conflict. We made sure to work on separate branches for each feature, and commit, push, and merge often.

AWS RDS
We hosted our development and test databases on AWS RDS. We encountered some problems with running over free tier limits, and had to move our test database to another account halfway through the project in order to not get charged. Setting up the databases and getting the right permissions were definitely the most tricky aspects, but things went smoothly after the initial setup.

AWS S3
We use an AWS S3 Bucket to store images uploaded by users. We initially ran into a lot of trouble with permissions to be able to access the bucket from our application. Once we got the environment variables and permissions set up properly, we were able to seamlessly link images to their corresponding listings.

Heroku
We used Heroku to host our application in production. The production database is also hosted on the Heroku instance. For the most part, Heroku was easy to work with and had useful integration with Github to automatically deploy when changes were pushed. One issue we ran into was building with both Ruby and Javascript with Heroku when we added node modules in order to use Bootstrap for some of our styling.

Code Climate and GitHub Actions
We used GitHub Actions to automate our testing. We originally ran into issues with how to use the environment variables, but we found GitHub secrets to be a solution. GitHub secrets essentially encrypts our environment variables needed to run our code which can be called in the GitHub Actions workflow yaml file. Whenever code was pushed or there was a pull request, GitHub Actions would install ruby and all the required gems. It would also set up the Code Climate test reporter then run our cucumber and rspec tests. At the final step, it would publish the code coverage by the simplecov gem to Code Climate.

Notable Ruby Gems
pg (postgresql) ➔ for our databases
cucumber-rails/capybara ➔ to test user-facing functionality and views
rspec-rails ➔ to individually test models and controllers
database_cleaner ➔ to clean database after test runs
simplecov ➔ to track code coverage and generate reports
google_sign_in ➔ to allow sign-in via Google OAuth
webpacker ➔ to allow usage of node modules for Bootstrap
aws-sdk-s3 ➔ to interact with S3 bucket for images

Repo Contents

For each entity in our database diagram, there will be a corresponding model in our repo under the /app/models/ folder. To gain a better understanding of our database, there is a schema file located in /db/ folder that contains the attributes of each table/entity and the foreign keys as well. The /config/routes.rb contains all the different endpoints that a user could access through the application. The logic for each route can be found in the /app/controllers/ folder. All html files can be found in the /app/views/ folder. It's organized based on the controllers so that rails can automatically map the controller functions to the proper view. The README.md details on how to start the application, how to run tests, and the contiguous integration information.

Links

Pivotal Tracker
GitHub
Heroku Deployment
Code Climate
Poster/Demo Video