Homelink Project Testing Report

Levon Brunson, Timothy Frank, Latifah Aljafar, Reagan Hunt

The Project Testing Report should (as a group) include a short summary of your testing plan and methodology (how you have tested various parts of your application, what software you are using for testing).

Testing was a very important part of our development process. While building various components and views we divided the work so that we could work independently and asynchronously, which meant that we had to be careful in making sure everything fit together when consolidating code.

This meant each of us would perform manual functional tests -- which for example in the navigation bar meant clicking each link, trying to navigate from every page to every other page, resizing, etc. -- basically ensuring that everything functioned as intended before pushing to the repository. We felt that automated functional testing wasn't necessary at the onset of the project since there weren't enough moving parts as of yet to warrant it. However, as of submitting this we are integrating the Cypress e2e testing framework into the project within the next few days before submitting our project in order to confirm there are no small lingering issues.

We would also perform manual integration tests, so whenever there were several large components coming together we would spend one of our check-in meetings going through the functionality of the website, noting any bugs or unexpected issues, and then adding them to our Asana board to keep track of them. Through this integration testing we picked up on a pretty important issue where new users that signed up would be unable to log in because of a mismatch in how their user IDs were being set in the Firestore.

The final component of our testing process was running vigorous unit tests, using Vue's built-in test utils. Under normal circumstances we would have incorporated these into the codebase from the get go, but given the shortened semester and front-loaded deadlines we had to move quickly to get our prototype ready in time. However, after that we found we had more breathing room and are combing our code top-to-bottom, adding unit tests for all of our non-trivial components.

It should also have an overview of what was discovered through testing.

There were plenty of small bugs that aren't worth mentioning, but in terms of larger issues: an issue with the signup modal where the user would be created but the page wouldn't redirect or update; a login issue where the user could log in but would be logged out immediately because the session wasn't being stored correctly; another login issue where new users wouldn't be able to log in because their UUIDs were being set and gotten in two different ways (an integration issue, as mentioned above); a router issue where certain routes could be accessed even when logged out; a payment issue where the stripe API is out of sync with our firebase project. The

API was fetching a one rent for all our tenants. It was fixed by associating the product firebase collection with the API checkout session.

Additionally, you should outline INDIVIDUALLY (but in the same document):

- What you personally did in terms of testing
- What tests were run
- What the tests showed
- What you did to fix any bugs or issues

Levon

I wrote and performed tests for the landing page, the sidebar, the unit page, and in tandem with Reagan worked on the dashboard. There were unit tests for these components, and they helped reveal some of the bugs listed above -- there weren't that many which were revealed by the unit tests though, these were more sanity checks than anything just because the project scope was so small it was easy to remain relatively bug-free because there were very few complex interactions. To fix the bugs I did a lot of console logging in order to pin down where something was going wrong.

Latifah

- What you personally did in terms of testing
  Unit Testing in the payment was limited, since we need an actual card to complete the scenario. Most bugs I faced were during development.

- What tests were run
  Unit testing on payment, payment details, tenant profile,tenant roommates, and tenant settings.

- What the tests showed
  Our test units were very simple , they did not show any bugs. Most bugs were found during the development phase.

- What you did to fix any bugs or issues
  My work was mostly on the tenant side, I worked on five components: payment, payment details, tenant profile,tenant roommates, and tenant settings. At the beginning of the project, I did not know how to fetch the firebase collections. And due to this reason, I faced a lot of bugs. The one thing that helped me overcome this challenge was using the console.log. Moreover, The ESlint was recommending that I should use arrow functions to create the vue methods. This recommendation messed up my implementation. I could not refer to the global vue using the keyword 'This' . The last bug I dealt with was the stripe API integration. I created the checkout vue component,but it was not linked to our

firebase collections. The easiest way to solve this issue is to create the product on the client-side script, and then link it to our collection.
However, this solution is not secure. The client can easily create the product and alter the price. so I needed a server-side solution. After a lot of searching, I found out that I can use firebase functions to invoke stripe API functions.

Reagan

I worked mainly on the landlord and tenant dashboards (home pages). This required me to become familiar with Vue, then create alert, todo and request components. To test these, I displayed sample data written in the Vue files. At this point, I had a lot of UI problems to tackle, from alert/request/todo items overflowing their containers to malfunctioning modals to missing icons.

My next task was creating mock data in the firebase real-time database, then attaching this to my frontend. Once the connection was established, I ran a long series of tests (actions) to make sure my delete functions were correctly deleting data from the database. At this point, while items added to the database were immediately rendered, and the delete function successfully deleted items from the database, the delete function was not visually removing items from the screen. I had to reorganize the content of my files to listen in real time for delete changes in the database and use those events to trigger frontend (delete content) actions. Once this was all straightened out, it was clear my components were successfully rendering data from the database.

I ran WAVE accessibility checker on my UI and tested the appearance in multiple browsers. I also added functionality to handle resizing of the browser, including reorganizing the page and omitting elements that were not absolutely necessary.

Timmy

I worked mostly on getting firebase integrated with the landlord units view and the landlord individual units view. I also integrated firebase for the authentication and registration of users. Of course there were a number of small issues that came up along the way, but the scope of this project was small enough such that it did not require significant testing. However, the tests I ran personally were testing logging in with both types of users (landlords and tenants) and making sure that different pages were rendering correctly. One issue that I identified was that if you went to the path for a landlord's specific unit while logged in as a tenant, you would be able to see that data. Of course there would not be a way to navigate through to that page through the normal course of the website, but if you had the link it would be accessible. To fix this, we used Vuex and local storage to guard access of components that should only be accessed by certain

types of users. I also tested different behavior of adding tenants to units and making sure that the page would update appropriately. At first there was an issue where you had to refresh the page before seeing the new tenant that was added to a unit, but then I fixed that by adding a componentKey to the div and using that to refresh it at a specific time.