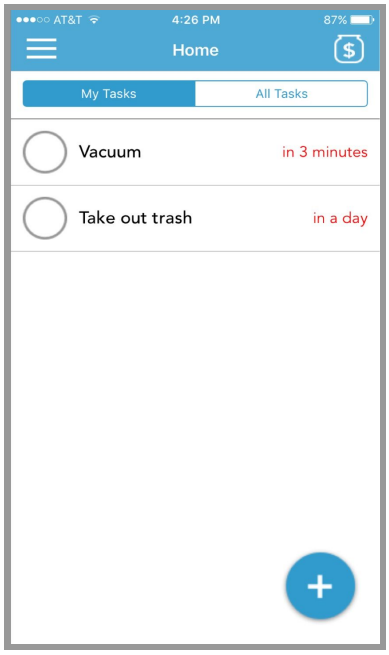
Jar

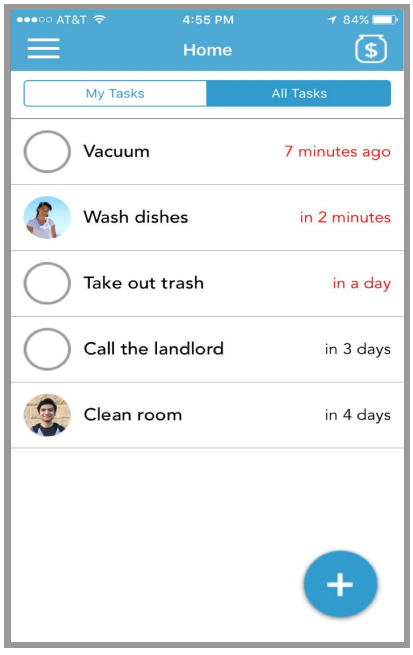
Motivating the completion of communal tasks, a dollar at a time.

Evan Lin - Software Engineering
Tessera Chin - Usability Testing
Michael Chung - Team Manager
David Morales - Design

Problem and Solution Overview

In the needfinding stage of this class, we found a common problem among housemates: people wanted an effective way to split responsibilities with their roommates. Jar provides a simple interface to distribute and manage household tasks, and motivates housemates to get things done. When someone fails to complete their task, Jar draws a small monetary penalty, which is collected in a virtual “money jar.” The house can then spend the money together on a group dinner, household furnishings, or whatever else they like. The idea is to balance negative feedback (the monetary penalty) with positive reinforcement through community building, in order to encourage communication and responsibility between housemates.

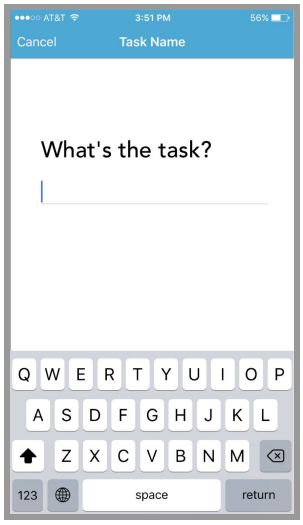


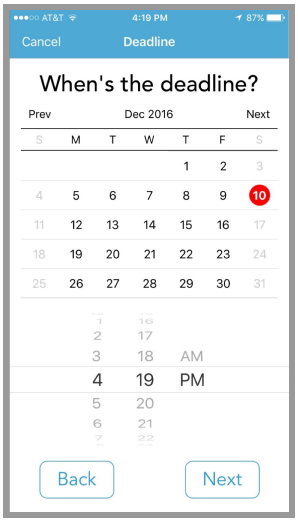


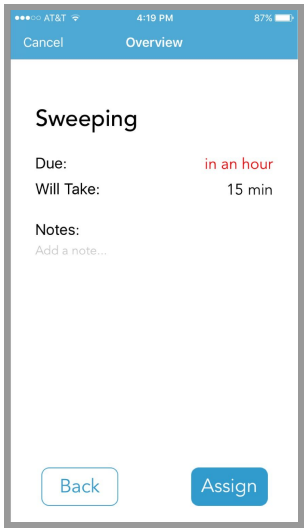
Tasks & Final Interface Scenarios

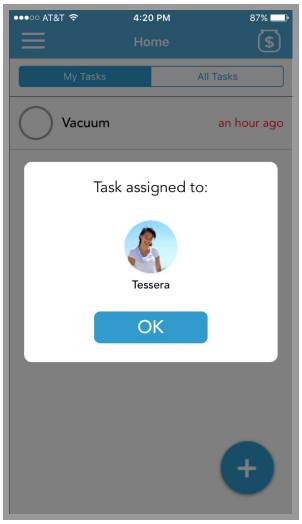
In order of increasing complexity, our three tasks were to: create/assign tasks, transfer tasks, and use the accumulated money the virtual money jar to buy something for the house.

Since Jar is a task manager, it only made sense to allow users to create tasks. This was our simple task, and we imagined that creating and assigning household tasks would be the basic use case of Jar.

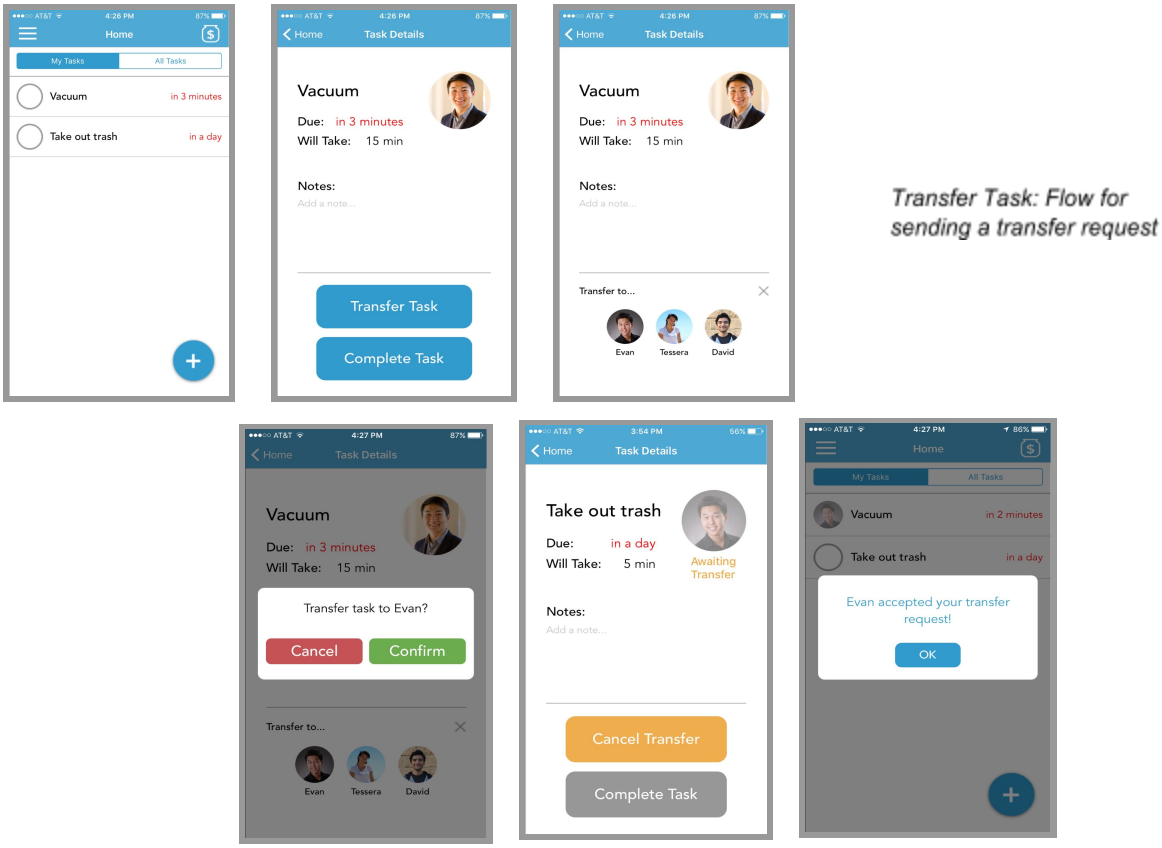




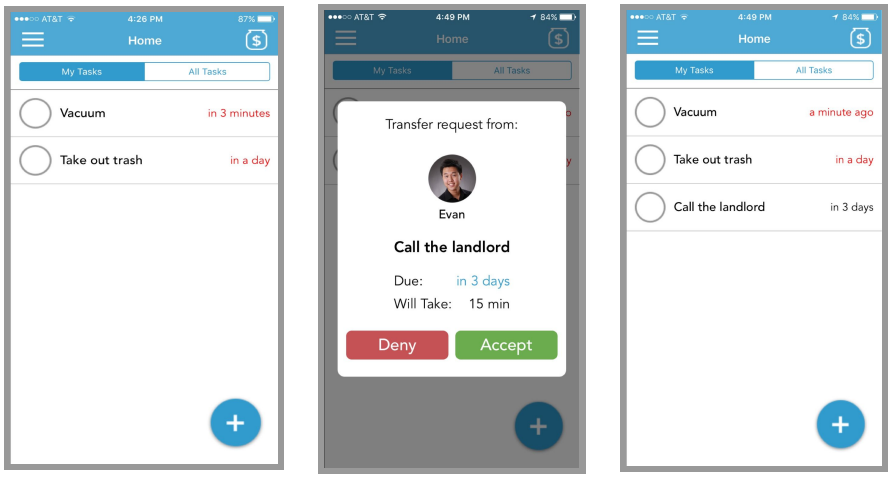




Our medium task was to allow users to transfer tasks. Since we saw Jar as an app that builds community, we wanted to ensure that there was some flexibility in task assignment; it would go against our values if our app's rules were inflexible and caused more problems within a household. As a result, we decided to include the functionality of letting users choose specific housemates to transfer tasks to. A transfer would only be successful if the other party also agreed to it.

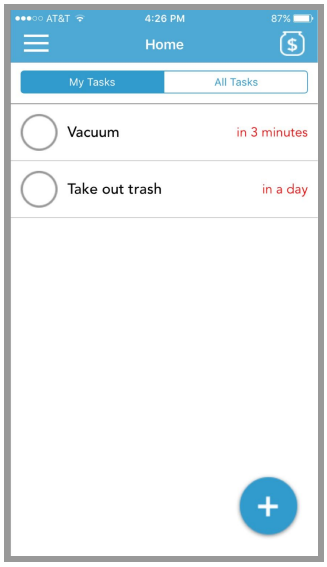


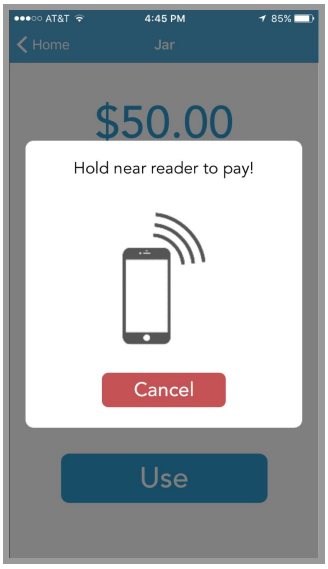
Transfer Task: Flow for sending a transfer request

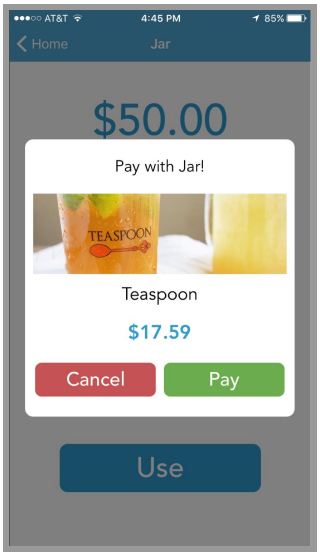


Transfer Task: Flow for receiving and accepting a transfer request

Finally, our complex task was to allow users to buy things for the house using the money accumulated from penalties. Monetary penalties would incur if users did not complete tasks by their deadlines, and the household could eventually use these funds to buy anything they want for the group, which includes food, household furnishings, group events, etc. Like we mentioned for our medium task, we want Jar to encourage a sense of community, and that's exactly the reason why we decided to include this complex task.



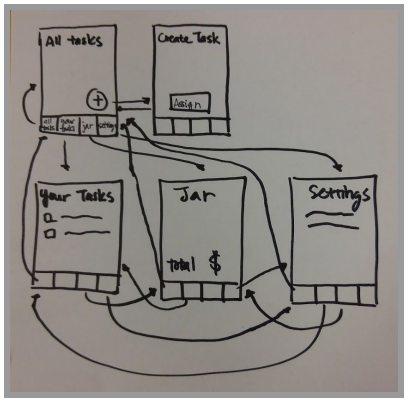


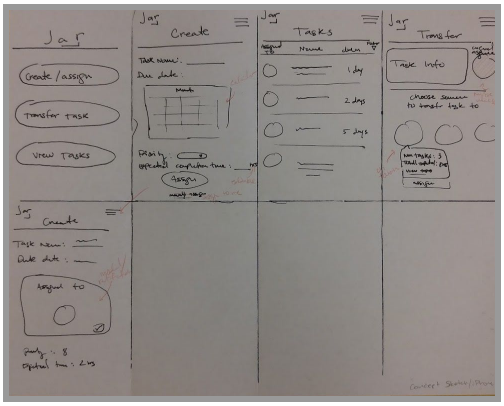


Design Evolution

Concept Sketches

Our initial design sketches included interfaces for Apple Watch, iPhone, and iPad. We quickly ruled out Apple Watch due to the extremely limited real estate. The iPad interface initially showed a lot of promise since we could show multiple views at once; which eliminated the need for the user to switch back and forth between screens. However, it eventually became more important to us that our interface remain as simple and clean as possible, which left the iPad design feeling too empty. On the other hand, the iPhone interface featured very simple interfaces and task flows that didn't appear overly cluttered. The need to switch between screens was outweighed by the more efficient use of space.

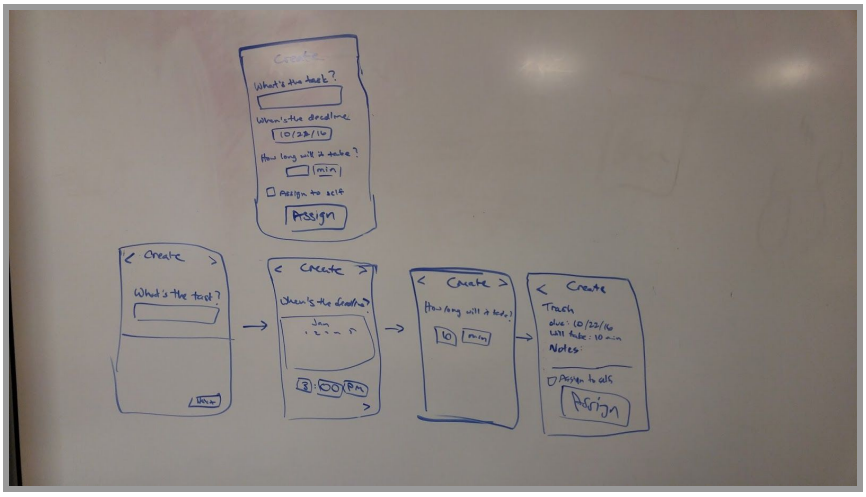




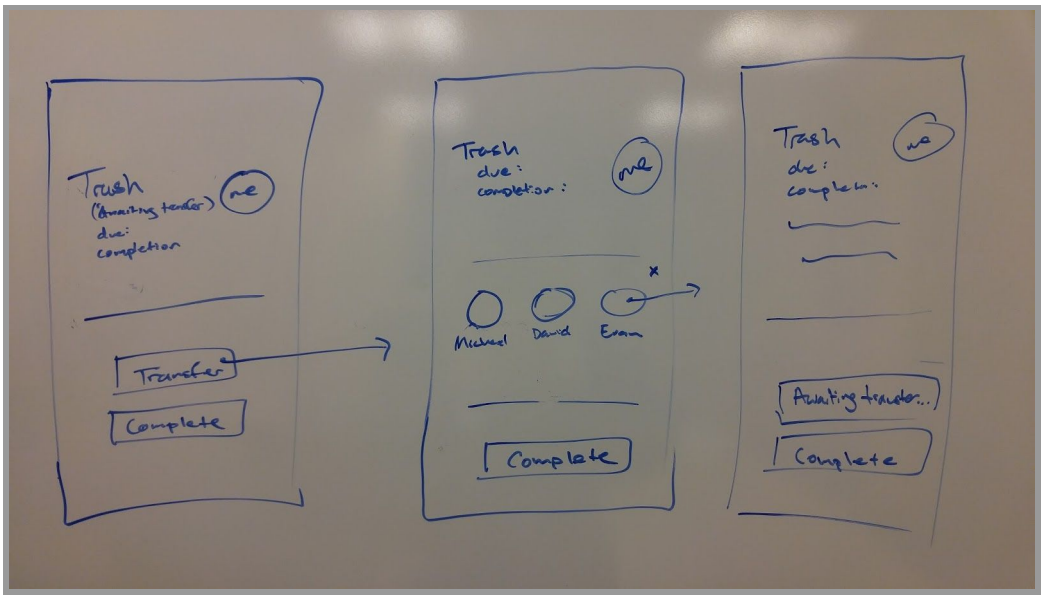
UI Storyboards

The UI storyboards for our task flows aimed to present all information in a logical manner, without overcrowding any single view. This goal came into play most notably in the 'Create Task' flow, when we had to decide between placing all fields on a single form or splitting the flow across multiple screens.

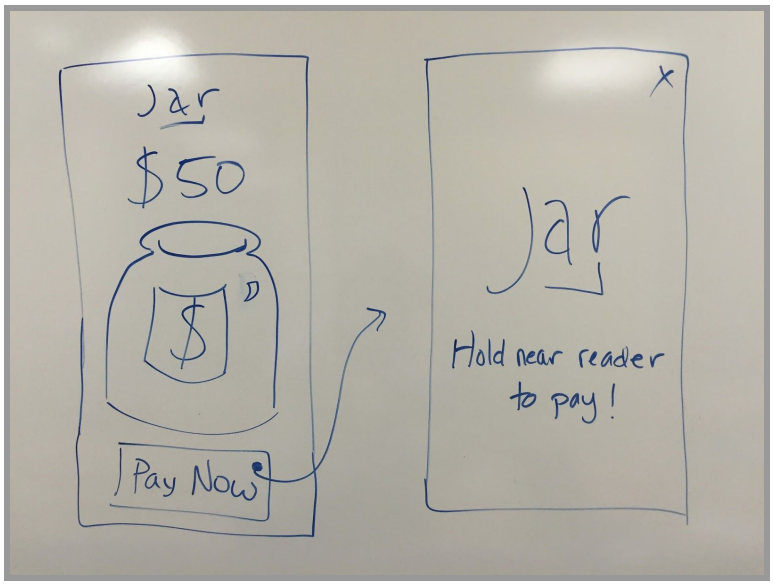
Create task flow:



Transfer task flow:

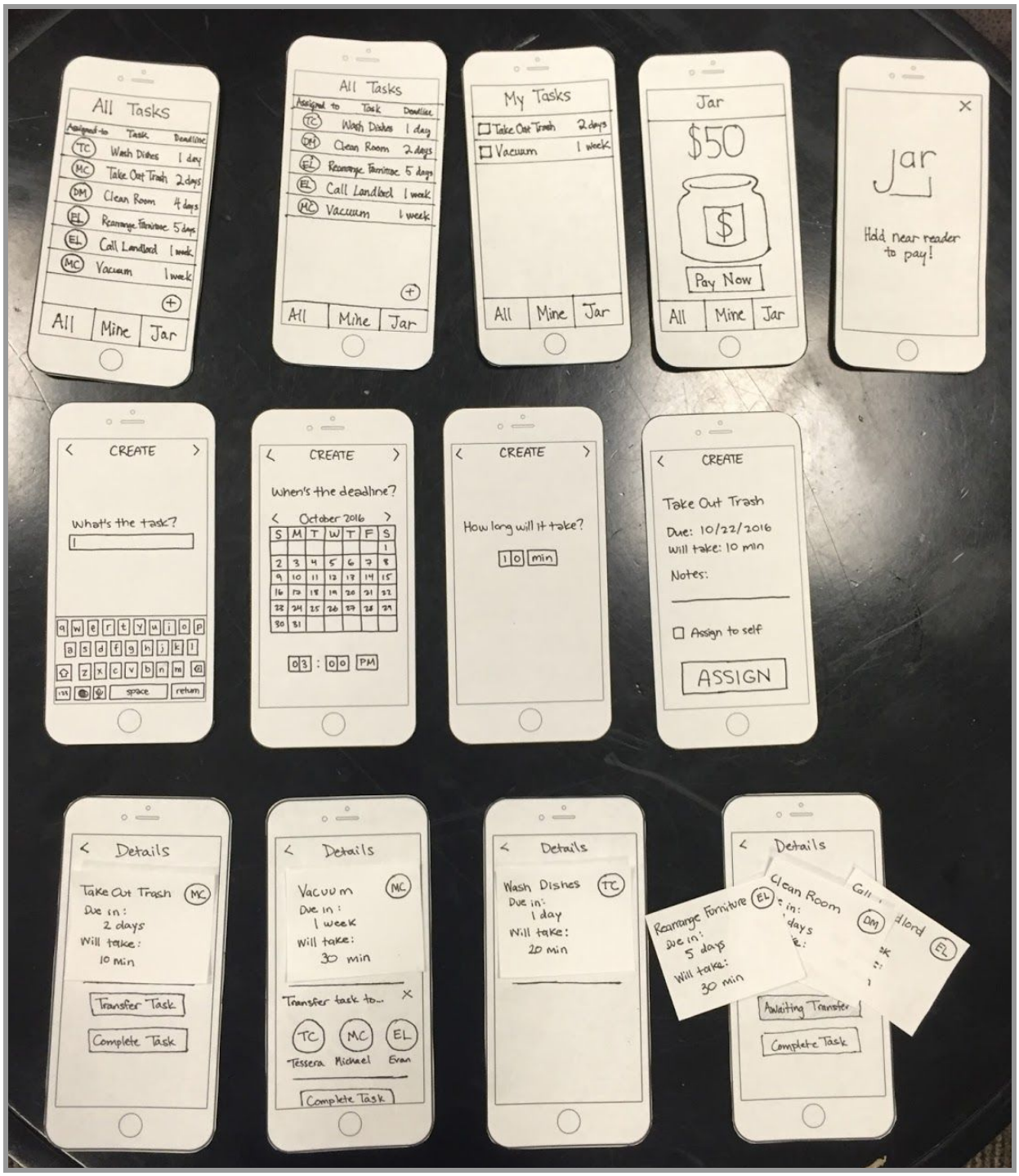


Money jar task flow:



Low-fi Prototype

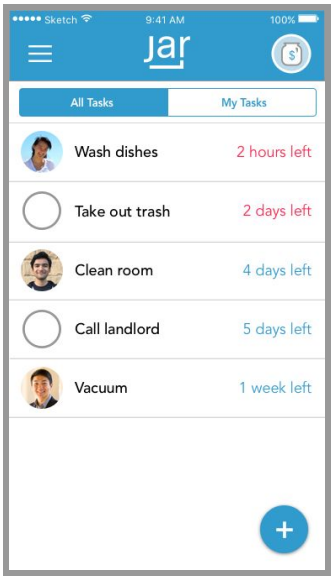
Our low-fi prototype was essentially a direct translation of the UI storyboards into more formalized structures. In addition, we fleshed out initial versions of the home ‘Tasks’ page, since it is the basis from which everything else stems.



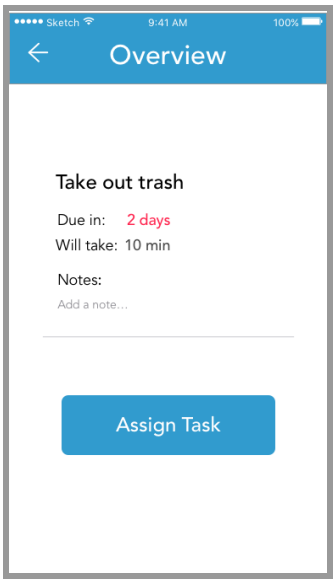
Medium-fi Prototype

The design for our medium-fi prototype was based off of our low-fi prototype, built in Sketch, and assembled in Invision. On a suggestion from some of our low-fi testers, we replaced the lower tab navigation bar with some more sophisticated tools such as a native Navigator for switching between main pages, and a Segmented Control for 'All Tasks' and 'My Tasks.' We also removed the 'Assign to self' checkbox in the 'Overview' page of the task creation flow, since it just ended up being confusing to users. Finally, we added notification modals throughout to provide the users with more feedback about the state of the app.

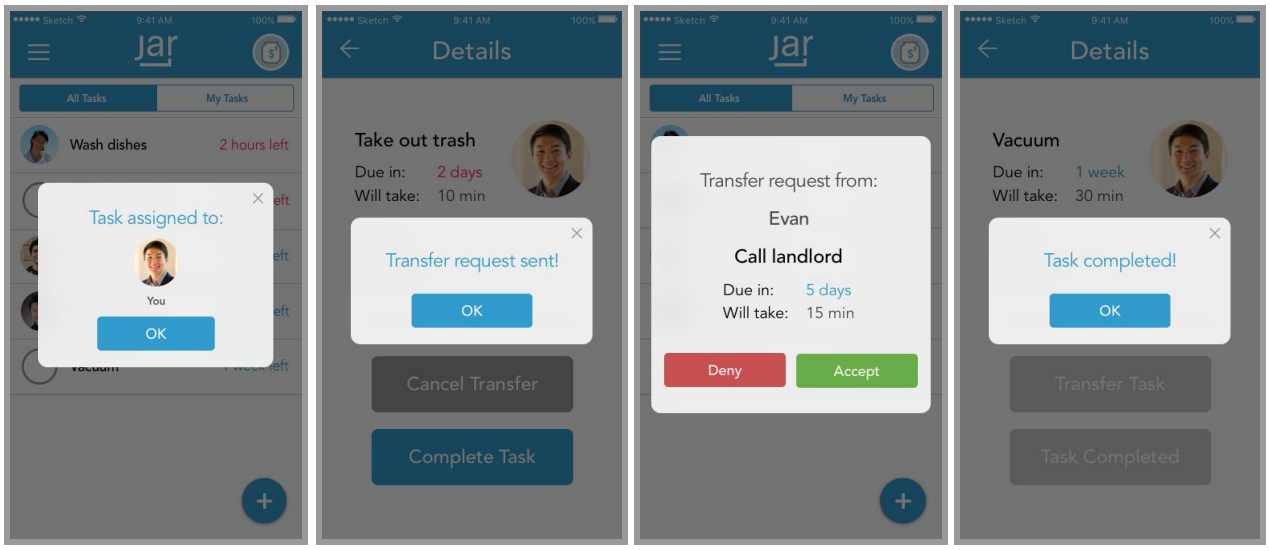
Navigator and Segmented Control:



'Assign to self' checkbox removed:



Notification modals:



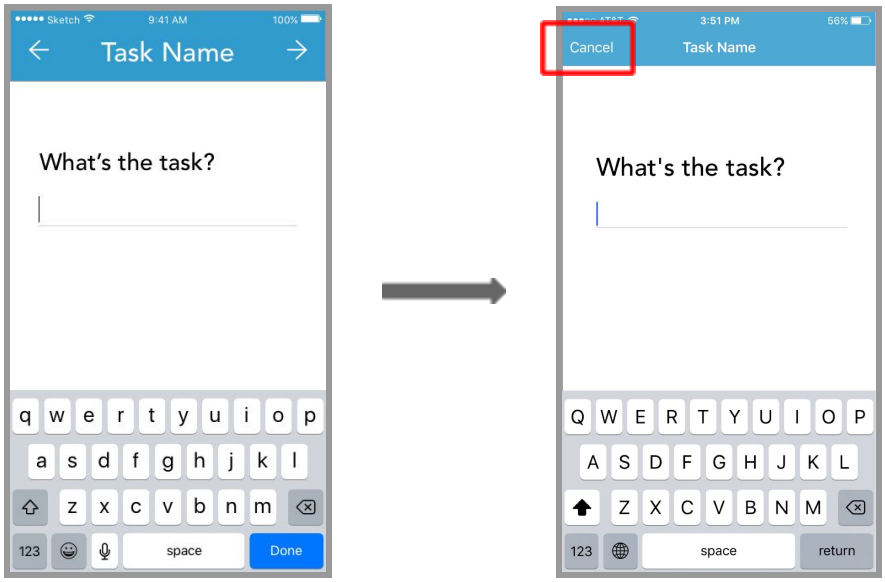
High-fi Prototype

Refer to the ‘Major Usability Problems Addressed’ section for design changes in the high-fi prototype, and to the ‘Tasks & Final Interface Scenarios’ section for images.

Major Usability Problems Addressed

H2-3 [Severity 4] - “The interface of creating/assigning a task doesn’t allow the user to cancel creating the task”:

We fixed this issue by adding a cancel button to the top left of each screen in the task creation flow.

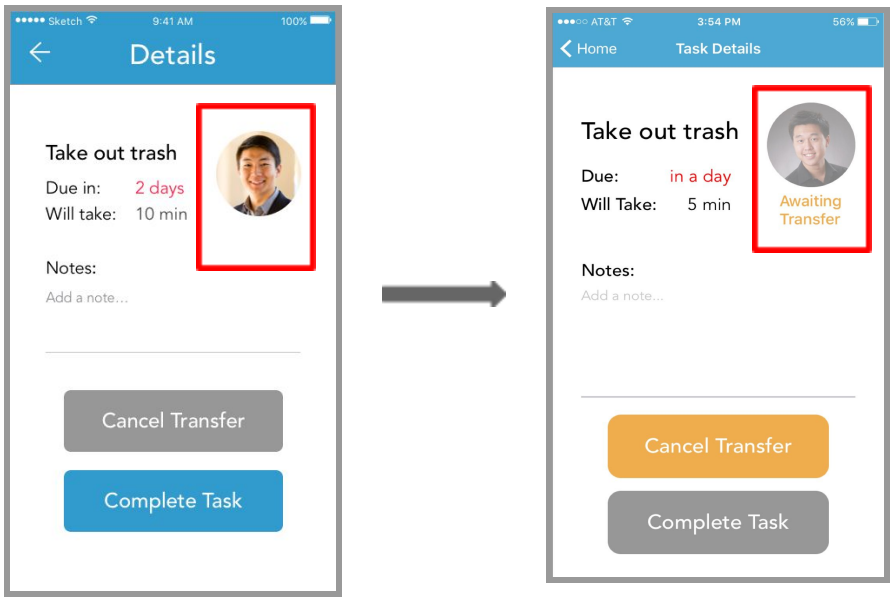


H2-7 [Severity 4] - “When the user reaches the end of the complete task flow before submitting and notices an incorrect input, he has to go all the way back to the corresponding page to edit…”:

There was a number of ways that we could have fixed this issue, such as making the fields editable from the task overview page, or navigating the user to the appropriate page when a field is tapped. Given the time constraints, however, we were not able to incorporate these fixes.

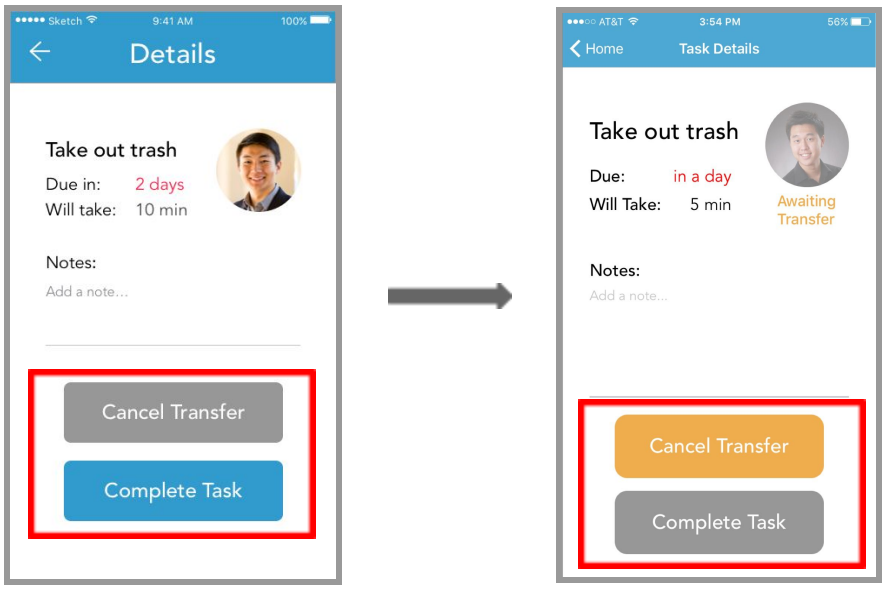
H2-1 [Severity 4] - “...there is no indication of whether the transfer request sent has been declined or the other user hasn’t responded yet.”:

To fix this issue and reflect the status of a transfer request, we changed the details page and introduced notifications. In the details page the image in the top right changes to a semi opaque image of the user who will deny or accept the transfer. Below this image is also text stating that the task is ‘Awaiting Transfer’. Once the other user responds to the request, a notification will pop up with the response and the image will be that of the new task owner.



H2-4 [Severity 4] - “After transferring a task, the “Transfer Task” button goes gray, which is the same action for when the buttons become deactivated after completing a task.”:

To fix this issue, we changed the “Transfer Task” button to orange whenever a task was in the process of transferring.



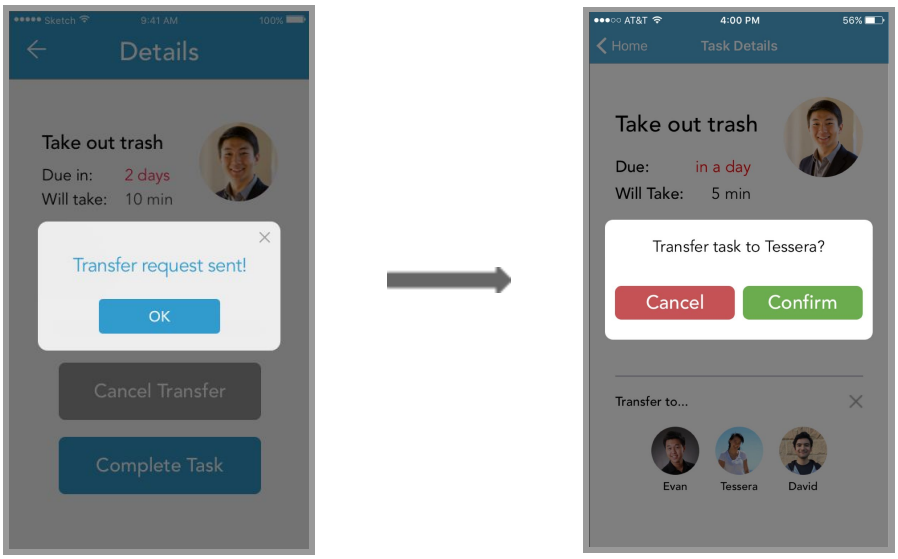
H2-3 [Severity 4] - “After creating a task, there is no way to edit or delete the task that you created”:

Considering that editing or deleting a task would require all other users to agree on the changes, we decided not to allow editing and deleting tasks. For the purposes of this class, we wanted to focus on a single user’s experience.

H2-4 [Severity 4] - “In the “transfer request sent” dialogue, there are a cross button on upper right corner and an “ok” button, both for acknowledgement purpose. If the user wants to cancel the transfer (“request” at the moment), he/she has to click “OK” anyway and click “Cancel Transfer” in the next interface. If the confirmation box only contains one button, then it can be designed as a pop-up message.”:

H2-5 [Severity 3] - “ After clicking on a person in the transfer task screen, the transfer request is immediately sent. User cannot cancel the request if the user clicks the wrong person by accident. Although there is a “cancel transfer” button shown after the request has been sent, it is better to present the user a confirmation option before he commits to action, because cancelling request and cancelling transfer should be two different states.”

The suggested fix for the first issue was to “remove the redundant cross button or adding an ‘Undo’ button to the confirmation box.” We kept the cross button, but we added an extra pop-up modal for confirmation with “Cancel” and “Confirm” buttons after selecting a person to transfer a task to, which fixed both the first and second issues.



H2-1 [Severity 3] - “The “Complete Task” button is a little bit confusing here since the task is already transferred to someone else.”:

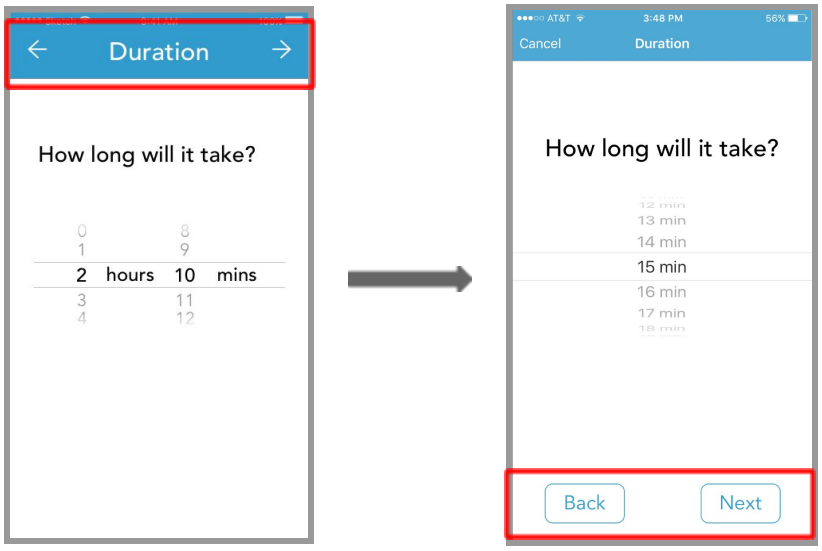
After transferring a task, we now disable the “Complete Task” button and gray it out, since it doesn’t make sense for the user to be able to complete a task while it is being transferred.

H2-3 [Severity 3] - “When a transfer request is received, the user will face a pop-up window with only two options: “Accept” or “Deny” before he/she can do anything. This is annoying because it is forcing the user to make a decision right away, and it increases the chance of error.”

We did not think this was an issue, so we did not make any changes to fix it. Given common users, it is very likely that if we gave users a chance to postpone decision making, they would not respond to any transfer requests, leaving the original user to wait indefinitely for a response.

H2-2 [Severity 3] - “When creating a task, the user is drawn from the top bottom to fill out information on each page, and then has to go back up to the top of the screen to move on to the next view, which is confusing and inefficient.”

We fixed this issue by moving the back and next buttons to the bottom of each view in the create task flow.

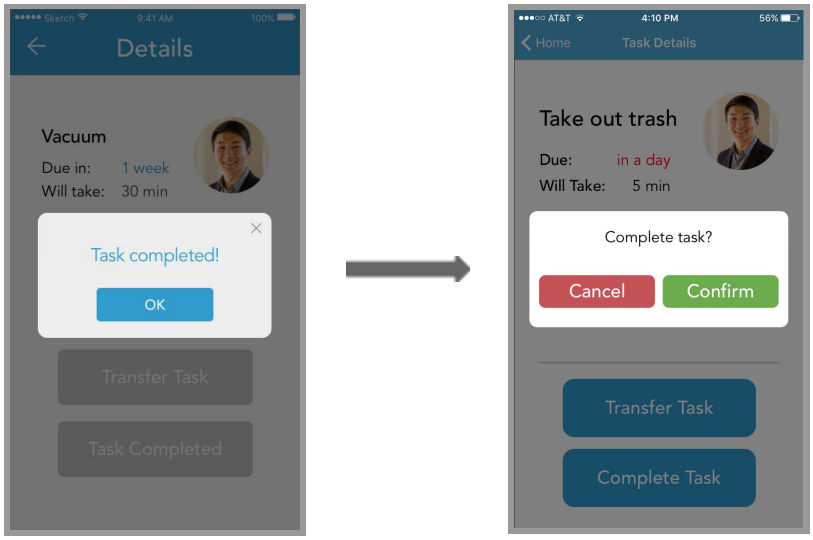


H2-7 [Severity 3] - “It would be annoying for the user to input information for the same common tasks each time.”:

Although there may be recurrent tasks that will be added to the task list, we considered that these tasks will have different deadlines and possibly different durations. In which case, it would make sense to just go through the entire create task flow. One simplification that we could have added was providing a list of common task names in the first screen of the task creation flow, so that the user would not have to enter a task name multiple times, but this seemed unnecessary.

H2-7 [Severity 3] - "...the user cannot unmark the task as completed from within the details page...If one wants to unmark a task because they accidentally pressed completed, he/she has to route back to the tasks view and uncheck the box next to the task.":

We decided not to allow users to unmark tasks as completed from the details page, but instead had them confirm that they wanted to complete a task. This would help prevent accidental completion of tasks.



Additional Changes

H2-5 [Severity 2] - "When creating a task, the "Deadline" and "Duration" panels allow users swipe left/right to move backward/forward in the process. However they both do not contain buttons to save/update the status"

All fields of a task during the creation process is automatically saved, so when users navigate back and forth between steps, any previously filled out field is redisplayed.

- create task flow: titles should remain 'Task Creation'

Prototype Implementation

Tools

Our high-fidelity prototype was built as a React Native app, using Exponent. React Native is a framework for building native mobile apps using JavaScript and React. It is a powerful tool for building a UI from declarative components. Coming into the project, none of us had any iOS or Android experience, so even though React Native was also new to us, we all had some experience with web and JavaScript.

Exponent is a set of tools built on and around React Native that helps developers in a few ways. It includes the Exponent XDE, a developing environment that runs React Native projects without having to go through XCode and allows developers to easily publish to and

develop on their own devices (or share with others). Exponent also provides additional APIs to JavaScript in order to avoid using native code.

Some limitations came with using Exponent as a tool, however. For example, Exponent doesn't allow you to drop down to native code, so a lot of lower-level customizations were not available to us. Additionally, Exponent is extremely recent, and still being constantly updated and improved. For example, we had to use Wizard of Oz techniques to simulate our notifications since we have no back-end to implement push notifications. However, Exponent just pushed an update adding an API for local notifications on December 7th, 2016.

Wizard of Oz Techniques

As mentioned above, we used Wizard of Oz techniques to implement our notification system. We used a special "gesture" (an invisible button in the bottom left corner of the home screen) in order to simulate a notification being received for a transfer request. We also implemented timers to pop-up modals when the deadline for a task passes.

Another Wizard of Oz technique was to imitate communication between multiple people using the app. For example, when a transfer request is sent using the prototype, there is no actual back-end to receive the request on the server, or forward it to another account. Instead, we simulate the transfer request being accepted by popping up a notification for "Transfer request accepted!" on the home screen three seconds after the request is sent.

The task assignment algorithm at its current stage in our prototype is also a Wizard of Oz technique. Instead of assigning tasks based on the actual amount of work people have done, since the prototype has no history of this, we just choose a recipient randomly from the housemates.

Hard-coded Data

We had to hard-code a majority of the data in our prototype. Any initial account setup, including bank account setup, is assumed to be already completed, and the user is using the app from Michael's perspective. In addition, the initial task list is hard-coded with information, deadlines, and assignments to other people in the house.

The functionality on the Jar page is also hard-coded. We used timers on modals again to simulate something like Apple Pay being used to spend money from the jar. Theoretically, the phone would obtain the transaction information from an Apple Pay reader, but for our prototype, we hard-coded a transaction occurring at Teaspoon for $17.59.

Future Explorations

In the future, we might like to implement a back-end for the app. Since most of its functionality is dependent on communication between different people's devices, the app would stay very much a prototype until this happens. There are also some features that we would have liked to implement, but weren't able to because of the time constraint. These included the ability to edit tasks after they are created, integrate people's task preferences into the assignment process, and having frequently-added tasks available in the task creation flow. We would also like to take advantage of some of the new libraries available in the newly updated Exponent SDK 12.0.0.

Summary

The final iteration of Jar presents a clean interface to manage communal tasks. From the beginning, we had a solid grasp of what we wanted the app's main tasks to be (creating/assigning tasks, transferring tasks, and dealing with the virtual money jar), and so the biggest changes we made were for UI purposes. After the iteration of each prototype, we collected feedback and incorporated changes into app, such that the task flows were all simple and intuitive. We are very proud of our final product, and hope that our hard work shows!