# GarageShare Final Paper

Matthew Hong, Adam Powers, Narine Cholakyan, Shimona Carvalho
March 20, 2012

## Introduction

GarageShare is an online service aimed at helping users share their possessions with their friends. While borrowing and lending is a common activity among friends, not many tools exist that help users manage this activity. However, we feel there are many aspects of this activity that could be aided by technology, which we will discuss. GarageShare hopes to make borrowing and lending easy and pleasurable, relieving some of the logistical and technical problems inherent in the practice. In the process, users get to save money, build social capital with their friends and communities, as well as reduce the consumption and aggregation of rarely used items.

### Concept

The recession of the last few years has given rise to a growing culture of simplified, minimalist and anti-consumerist living. Countless articles, television shows and websites dole out advice on decluttering your life and reducing needless purchase and consumption of goods. One simple and obvious way to reduce clutter and rampant consumerism is the simple act of borrowing an item from a friend instead of buying it. The gain to the borrower is obvious; avoid spending money and be able to return the item when done using it. The gain to the lender is more subtle but compelling nonetheless. The lender has an opportunity to help out a friend at minimal cost to themselves, as well as an appreciation that their rarely used item has come in handy for the friend. In addition, both parties can feel a sense of satisfaction in making an eco-conscious decision to reduce environmental waste by not purchasing another newly manufactured item.

For users to effectively borrow and lend items to their friends, one of the first pieces of necessary information that is not immediately available is whether or not a friend has the item one wants to borrow. Without prior knowledge of this, users need to contact all of their friends to request the item. More than likely, users contact just a few people and upon failure to locate the item, decide to purchase it instead. GarageShare can easily improve this process by providing an inventory where users can list items they are willing to share with friends, and allow potential borrowers to search their friends' inventories.

Another aspect of sharing that is problematic is the tracking of shared items. It is often the case that a borrowed item is forgotten and never returned. Here again, if the borrowed item is requested through GarageShare's inventory service, it is trivial for GarageShare to mark the item as borrowed and track the status from both the borrower's and lender's perspectives. Immediately, this is a huge cognitive offload for both parties as they no longer have to track this in their memory, but instead can rely on checking their page online to see the status.

This leads directly to the next issue that is the tracking of the return process. There is usually an expectation of when a lender wants their item back or when the borrower promises to return the item. If this information is entered into GarageShare at the time of lending, it can easily remind the borrower as the date approaches and even allow the lender to trigger reminders.

We believe users are more likely to need this kind of service for certain classes of items. For e.g. items that are used once in a while and stored the rest of the time are excellent candidates, as are items that are expensive or bulky. From the borrower's point of view, items that are expensive are good candidates as the borrower may not want to spend the money on a single use, or perhaps they simply cannot afford the item. Similarly, items that are bulky or large are often borrowed, since people living in smaller spaces often cannot afford to store such items, and would prefer to borrow and return them. Small, cheap items are less likely to be borrowed, as neither the borrower nor the lender have much motivation to track such a transaction. We aim to ensure GarageShare will help its users manage the types of items that are more likely to be borrowed and lent and more likely to require tracking.

Thus, we believe there are several aspects of the sharing process that could be aided with a service such as GarageShare. In order to test our intuitions and personal experience, we decided to gather user data through an online survey.

# Design Process

## Survey

To understand the needs of our target user base, we conducted an online survey across various social networks, such as Facebook and Google, and a total of 42 people responded to our questionnaires. Multiple choice questions were used to help quantify objective categorical responses, as well as open-ended ones, in order not to constrain the expressiveness of the responder's subjective evaluation of each subject matter. The questions we asked are as follows:

1. What kinds of items would you feel comfortable loaning to your friends?
2. What kinds of items would you be interested in borrowing from your friends?
3. What factors do you take into consideration when deciding whether to loan an item to a friend?
4. When searching for an item, would you be interested in knowing the location of the item?
5. Would you like to know the location of the items lent to your friends? If so, why and how often?
6. What was the item borrowed?
7. What did you like or enjoy about the transaction?
8. What did you dislike or have trouble with during the transaction?
9. Did you use any tools to help you with the transaction? If so, can you describe how and how well they worked for you?
10. What was the item lent?
11. What did you like or enjoy about the transaction?
12. What did you dislike or find difficult about the transaction?
13. What kinds of items would you feel comfortable loaning to your friends?
14. If you did use a tool in the question above, can you describe how and how well they worked for you?
15. Did you use any tools or services to help you handle the transaction?
16. If you did use a tool in the question above, can you describe how and how well they worked for you?
17. Did you use any tools or services to help you handle the transaction?
18. Any other thoughts on what information it would be good to be able to search on?

We first had a chance to look at the responses to the multiple choice questions since they are easily quantifiable and thus can be visualized in a simple 2 dimensional line graph. Several interesting results and patterns emerged, the first concerning the number of borrowers and lenders. We originally predicted that there would be more borrowers than lenders because, in terms of the imagined net profit, borrowers have more to gain than lenders.

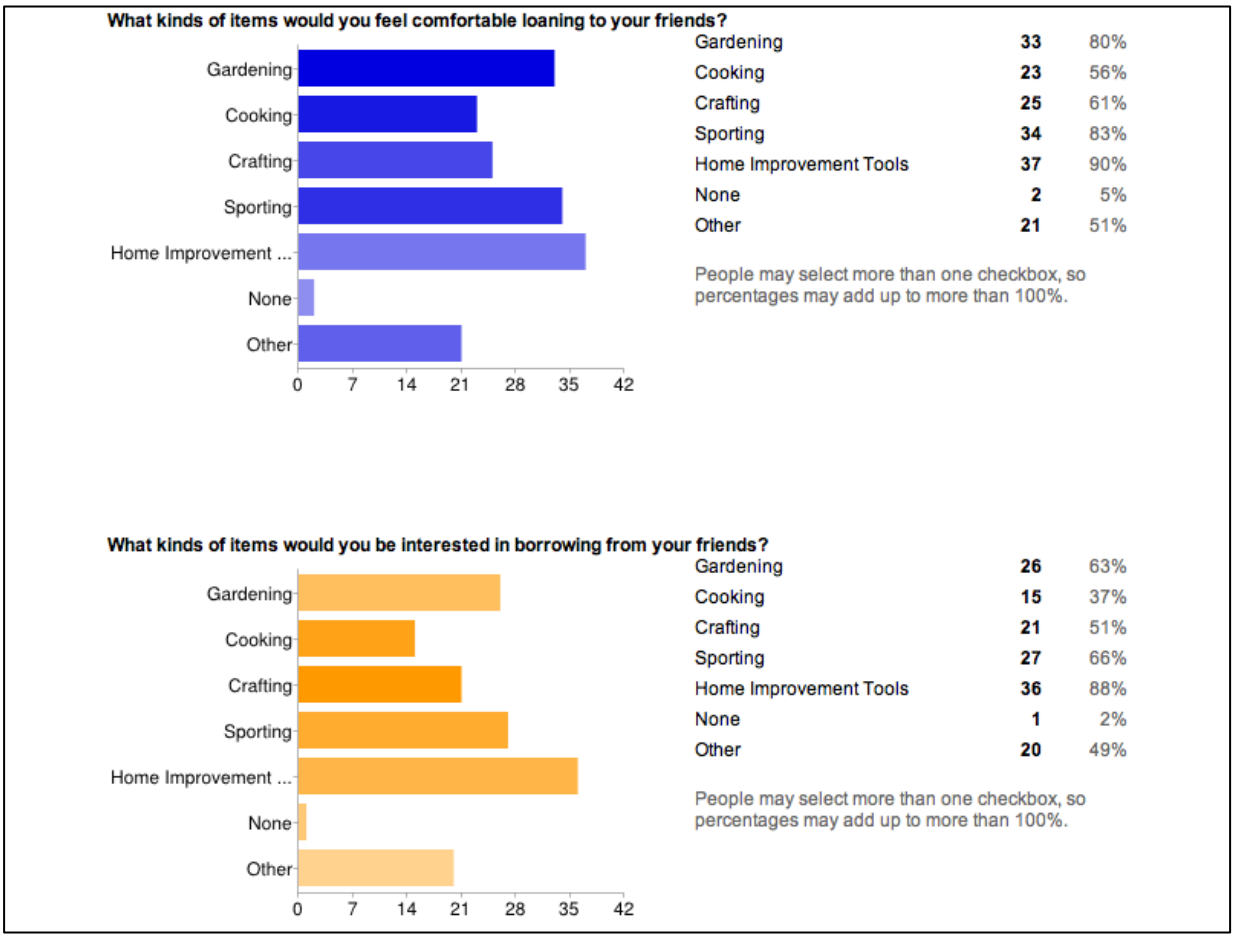


Figure 1. Types of Items

However, as the graphic depicts in Figure 1, and to our surprise, there seemed to be a fair distribution of the number of borrowers and lenders. As a matter of fact, there were slightly more people willing to share their items with friends. Also, we found a new, unexpected category of items that many of our respondents were willing to share, i.e. textbooks.

We also asked a question regarding location awareness to understand how people felt about sharing their location information, as shown in Figure 2.

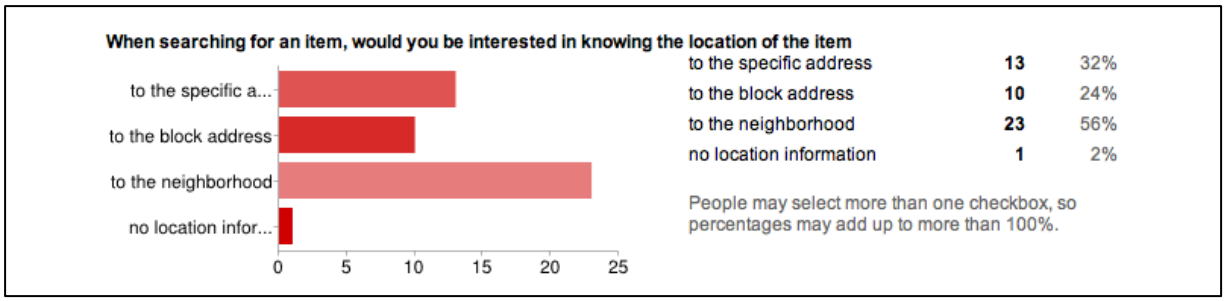


Figure 2. Location Awareness

Also somewhat unexpected, a majority of people were not as interested in knowing the exact location of the searched item, but were happy with the general region of the item. Whether people are being generous are not, this data will surely have a great influence on the way we should design map interfaces for websites that support sharing of goods.

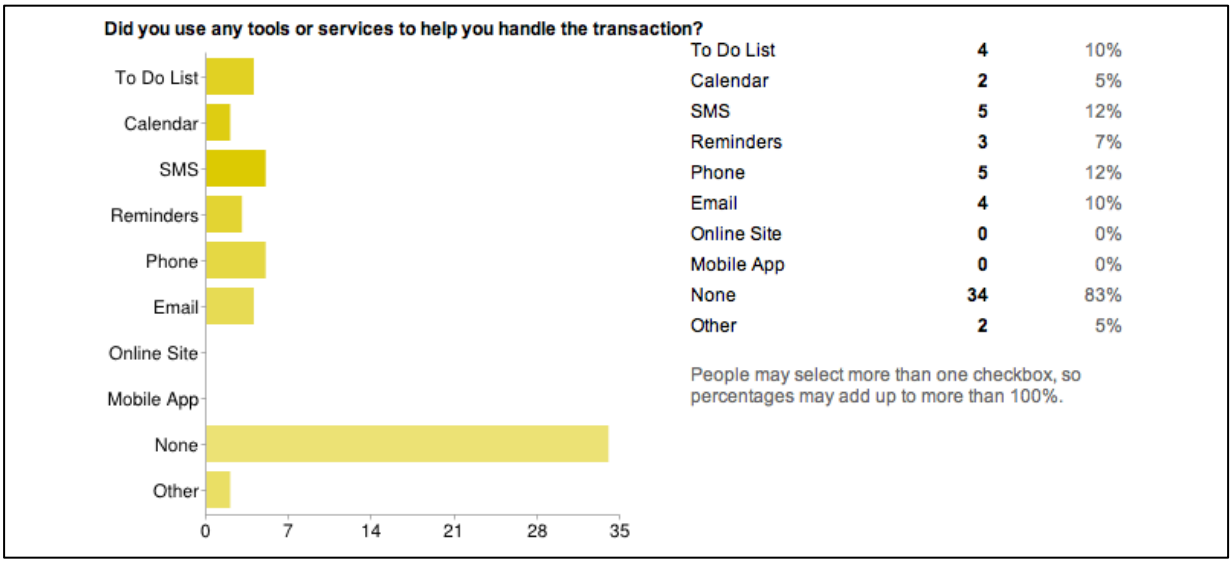


Figure 3. Tools and Services used by Borrowers

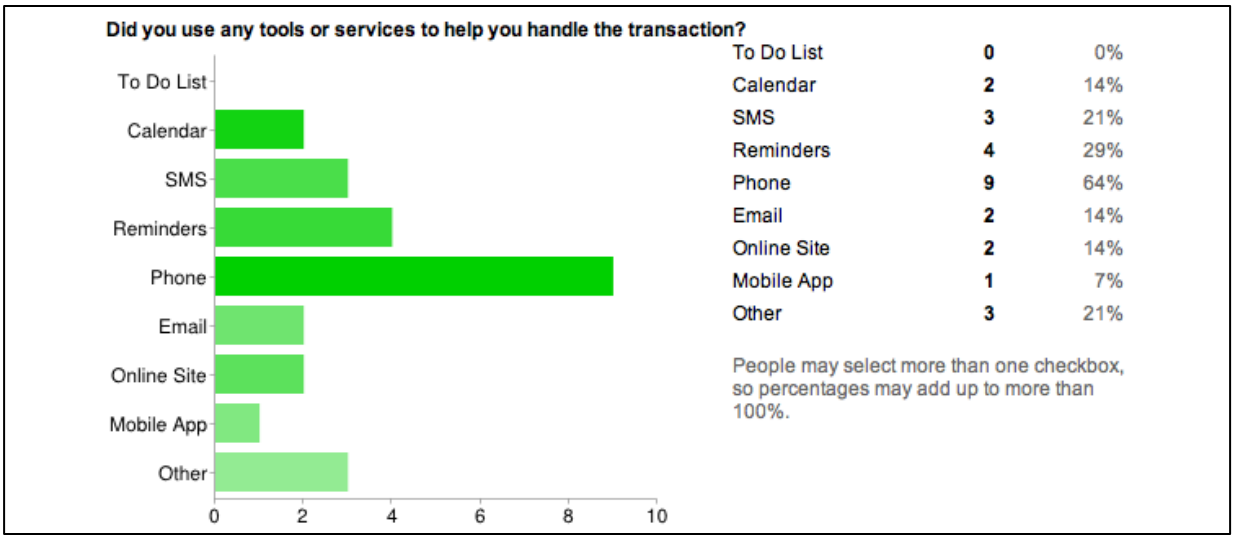


Figure 4. Tools and Services Used By Lenders

Another finding that left a lasting impression on us was the result of two parallel questions regarding the use of tools and services as a way to remind one's self of the transaction. As you can see above in Figure 3 and Figure 4, the borrowers' did not use many tools to handle the transaction. On the other hand, lenders were more concerned with the status of the lent item, and a majority report using phone calls to retrieve their items.

Thus our major findings were that both borrowers and lenders are interested in sharing items, and there are not many tools and services that are used to help the process. The types of items that people were interested in sharing matched our original impressions and suggested a new category - expensive textbooks. The quantitative data, supported by the subjective open-ended comments, showed that lenders were especially concerned with the tracking and return of their items, and the borrowers less so. This is one gap where a service could help provide lenders with a sense of control over their items, and remind the borrowers to return the item. In general, the results supported our initial concept and provided richer data from which to draw ideas and inspiration for aspects of the service.

# SITE DESIGN

The survey results pointed us to a dashboard design that offers users a sense of control over their sharing practices with minimal drilling through menus to access the most important information. Our original wireframe of our solution is shown in Figure 5. The wireframe shows the dashboard, which is a simple three column layout with multiple components in each column.

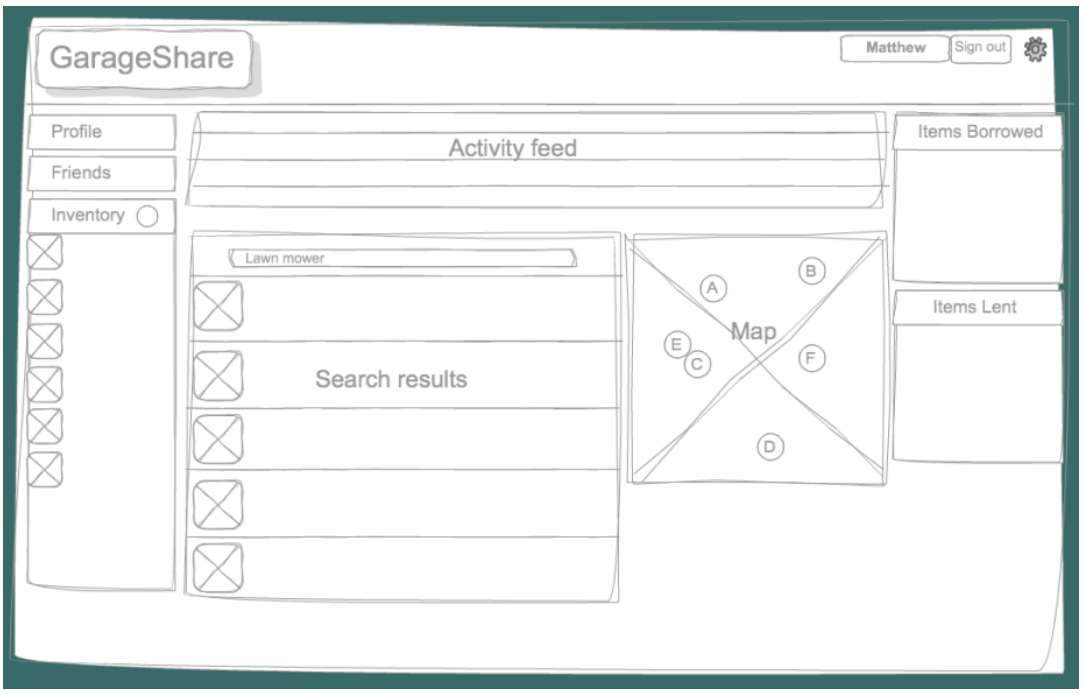


Figure 5. Dashboard Wireframe

## TECHNOLOGIES

The layout of the site and components was aided by the use of the Twitter Bootstrap framework which provides CSS that has been tested against browser incompatibilities to help layout and style various components of HTML.

Since the time available for completion of the project was limited and the goal of the course was programming for HCI, we decided the front-end was a higher priority than back-end server side code and databases. We focused on using HTML, CSS and Javascript to implement the functionality and front-end interactivity that we needed. In addition, jQuery UI was used for the modal windows, and Google Maps API for the mapping functionality.

All components were programmed in Javascript and jQuery with the taffyDB library being used to easily search JSON data in a query style format. Thus, the front end is essentially a functional prototype that could eventually be hooked in to a back-end. Since the JSON data files are currently only read by javascript, there is no actual update done to the databases and all data was created and loaded beforehand into the JSON files.

Although our final interface for GarageShare does not include Facebook integration, we did work on its implementation during the development process of our project. Facebook integration is based on Facebook's Open Graph API which would allow GarageShare users to login using their Facebook login information thus allowing GarageShare to access their basic information such as Facebook username, userID, email, profile picture and friends lists.

We started the integration process by creating a GarageShare application using the Facebook Developer App. However, despite various attempts to integrate with Facebook, we continued to run into obstacles and decided to deprioritize it. Instead, we chose to focus more on developing a user friendly interface and integrating the individual components of the dashboard into a cohesive front-end.

One challenge with the dashboard design was the need for multiple members of the team to be developing simultaneously on the same page. To help with this, we initially used separate HTML files for each of the components as it would allow us to edit each part independently and still be able to see the dashboard page as a whole. To do this, we had .load() functions that would populate each component. However, this caused new issues due to the asynchronous nature of the load calls. We were forced to use a breakpoint right after the load calls to ensure all HTML loads were complete before the Javascript began populating the components. This was ugly at the time, but allowed us to develop in separate HTML files and just pull them into the main HTML at the end when they had solidified.

As everybody in our team had different skill levels with the technologies we used, we were all able to work with different aspects of the site and gain useful technical skills during our experience of developing GarageShare.

## Components

We wanted the dashboard to provide components driven by the users' needs. Thus, the dashboard, seen in Figure 6, contains tracking components for borrowing and lending, a search component for finding of items, and an alert component to remind users when their borrowed items are almost due. In addition, we added a component for viewing friends' profiles and recently added items.

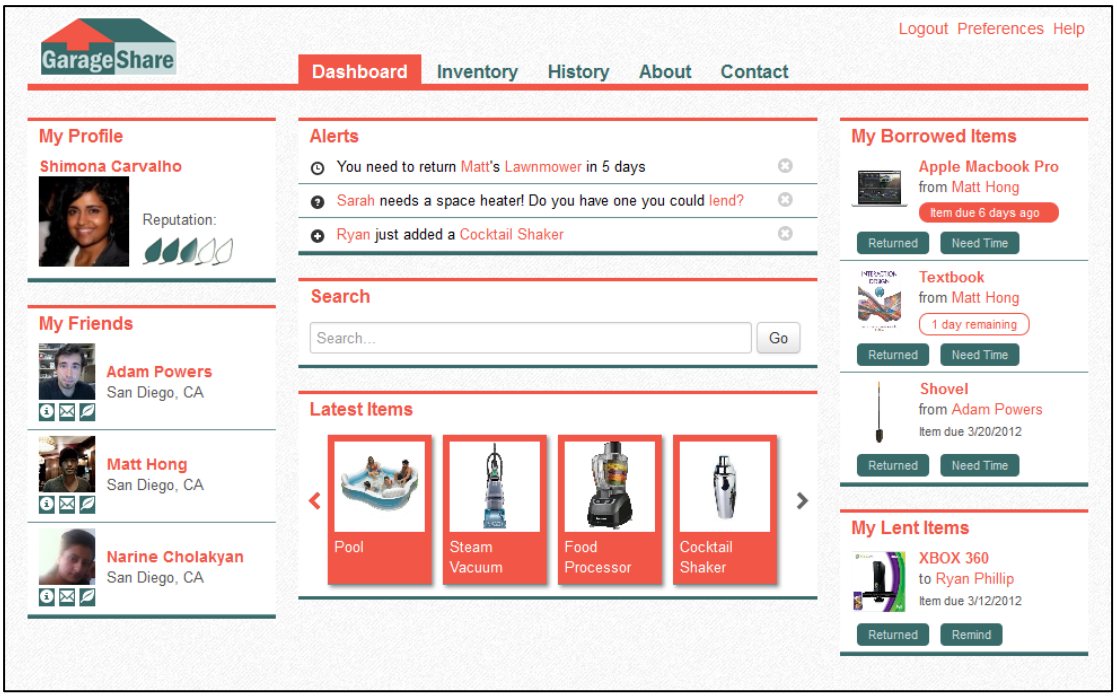


Figure 6. GarageShare Dashboard

## *Tracking*

The "My Borrowed Items" component shows a thumbnail and name of the item, as well as the name of the owner of the item. A date tag shows the date when the item is due. However, if the item is due shortly, i.e. within two days, the tag says "Item due in x days" rather than the date, and is bordered in red as well. This serves to emphasize the alert and gain the borrower's attention. Furthermore, when the item is past due, the date tag is highlighted red and is a more direct request for attention and action. This is one step toward bridging the gap where the borrower is less concerned about the return of the item than the lender is.

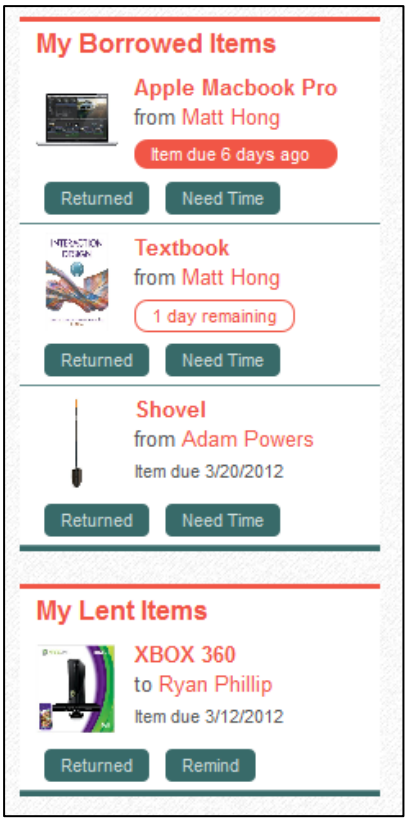


Figure 7. Tracking Components

Buttons allow the user to either mark the item as returned, an action that should be confirmed by the lender, or request more time for the item. However, these are not actively hooked in to a backend at this time. Clicking on the item thumbnail itself pops up a modal window with a more detailed item description as well as a link to a similar item online.

Providing a link to a similar item online makes the task of entering items easier for the owner, as the online item is likely to have descriptive specification information, such as size, features, capabilities, that could prove useful for the borrower without the owner having to enter it in manually.

The "My Lent Items" component is similar to the "My Borrowed Items" component except that it shows items that the user has lent to other users. The buttons provided allow the user to mark the item as returned, or send a reminder to the borrower. Again, these are not actively hooked in to a backend.

## *Search*

The "Search" component allows users to search the database of their friend's inventories to find an item they wish to borrow. Currently the search component searches all items, however the final product would limit the search to items belonging only friends who are willing to lend to the borrower. The search component slides out a map when a search is completed with markers showing the borrower's home, and the locations of the items. The map pans out to ensure that all of the available items to borrow are visible.

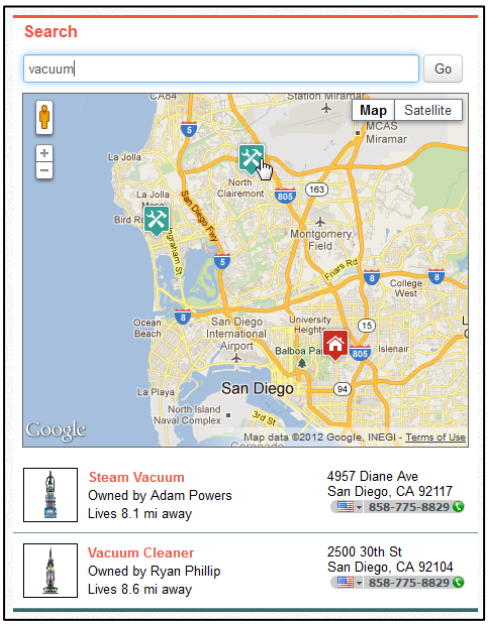


Figure 8. Search Component

To allow the user to connect the markers on the map with the results of the search, a two way linkage is provided. On mousing over the item in the search list, the marker associated with that item will bounce on the map. On mousing over the marker in the map, a tooltip will appear letting the user know which item is available at that location, who the owner is and how far away the location is from their home. This allows the user to browse either the list or the map, without having to constantly switch back and forth between the two.

The item list contains a thumbnail, the owner's name, address and phone number for easy access. Finally, clicking on the item will allow the user to view more details, evaluate a similar item online, request the item online, etc. in a modal window as shown in Figure 9.

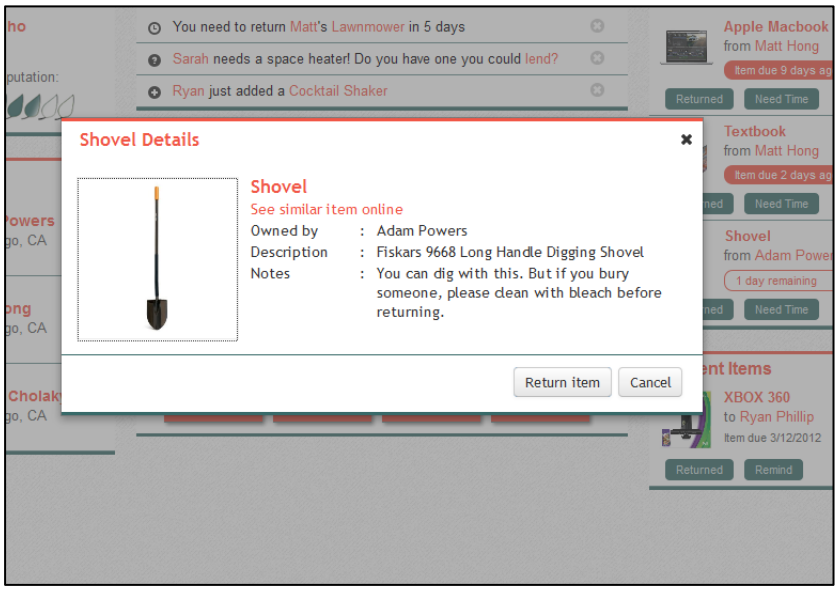


Figure 9. Modal window with description of items

## *Other Components*

The 'Friends' Component shows a few of the user's friends with buttons to message them, view their profile etc… Currently, the button functionality is not implemented, but we felt that addition of this component allowed users to 'see' the friends that they could help by participating in the site and encourage them to add more items to their inventory. The pictures of friends adds a human element to a dashboard that otherwise may resemble a

shopping site. It reminds users of the community and social capital aspect of GarageShare. It also gives users' a needed sense of comfort that they are sharing these items with friends they trust.

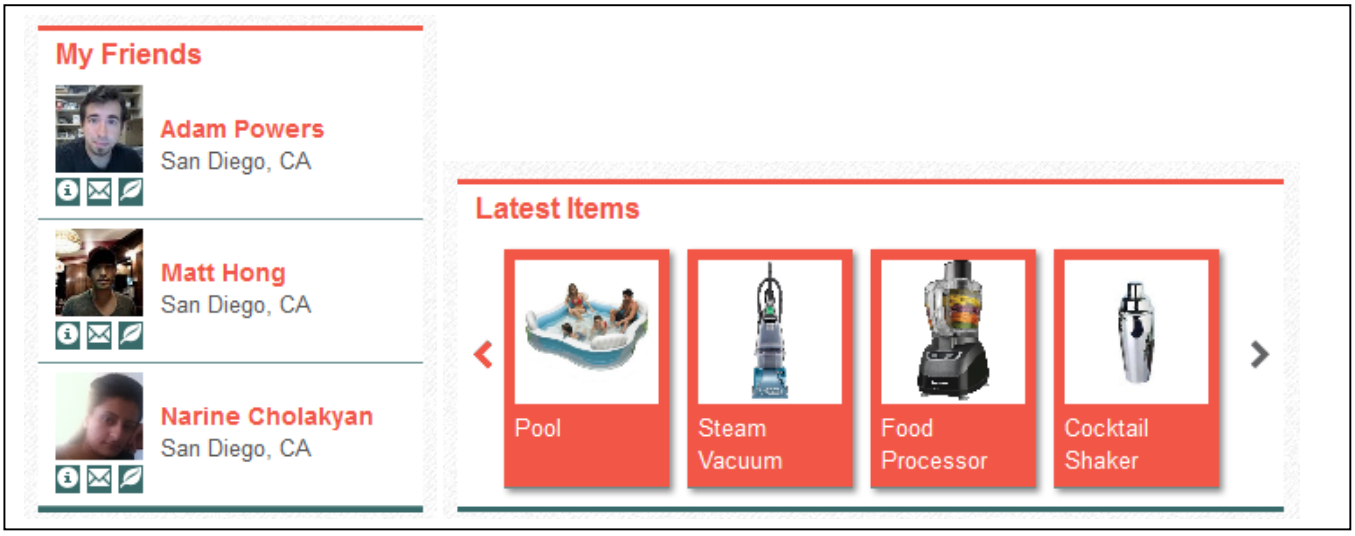


Figure 10. Friends and Latest Items Components

The 'Latest Items' Component simply shows the most recently added items within one's friends. One first landing on the dashboard page, this ensures that the front page is likely to change and have some new content when users login, thus bringing them back. It may remind them that they needed to borrow a particular item or even that they have a similar item they could be adding to the inventory to share with others.

## Login Page

To introduce users to the concept of our page, we also created a landing page, as seen in Figure 11, with a clear description of the service. Simple blurbs introduce the ideas behind the service and benefits of using the service.

The landing page incorporates the sign-in and registration as well, thereby allowing current users to login immediately with too many extraneous clicks. Registration is provided through the form of a modal window that validates user inputs on the fly with helpful messages, reducing the frustration associated with most online forms. This is shown in Figure 12.


Figure 11. GarageShare Landing Page

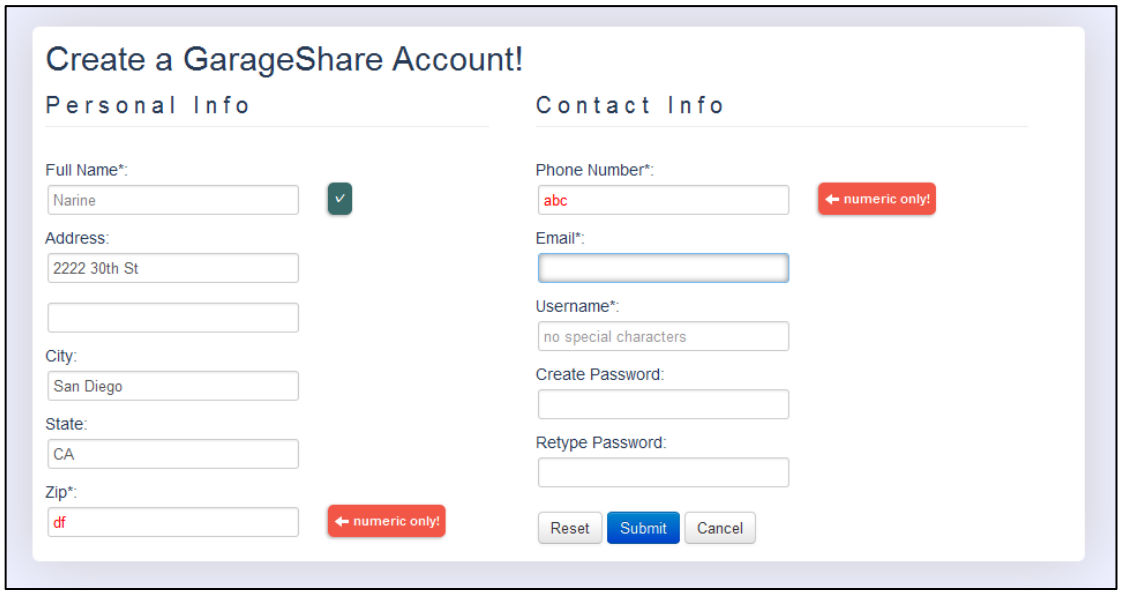


Figure 12. GarageShare Registration

# Code Documentation

The code is organized into folders as shown below. HTML files are at the top level, with js and css files within their own folders. The js and css folders themselves contain a lib folder for outside libraries. Further description of the main code follows this quick summary document tree.

- GarageShare
  - css
    - lib – Contains libraries related to Bootstrap, LESS and jQuery UI
    - main.css – Contains main top level CSS code, includes less variables
    - form.css – Contains CSS for login and registration form
    - shim.css – Contains CSS for components and modal windows on Dashboard
  - img – Contains images used on site
  - js
    - lib – Contains libraries related to Bootstrap, LESS, TaffyDB and jQuery UI
    - main.js – Contains top level javascript
    - prof.js – Contains javascript for setting up the User Profile on the left sidebar
    - narine.js – Contains javascript for New Account form validation and modal window creation
    - shim.js – Contains javascript for Search results, Mapping, Borrowed and Lent Items, New items, Alerts, Item detail modal dialog
    - item.js – Contains JSON data for items
    - user.js – Contains JSON data for users
    - trans.js – Contains JSON data for transactions
    - msg.js – Contains JSON data for alerts
  - pic – Location of User Profile pics
  - Login.html – Login page HTML
  - Index.html – Index page HTML
  - navbar.html – HTML for navigation bar, loaded dynamically into index.html

# JAVASCRIPT FUNCTION DOCUMENTATION

## *MAIN.JS*

**getCurrentUserId()**

Gets the current user id from GET string, which can then be used to personalize the page for the current User

**initializeDBs()**

Initializes the databases that are used by each component by loading the JSON file into a TaffyDB object. This DB object can then be queried using TaffyDB's functions.

**loadModal()**

Sets up modal windows using jQuery UI functionality. This just creates the modals for future use, it does not display them

**setupCurrentUser()**

Calls getCurrentUserId to get the id and then gets user information for current user.

**document.ready()**

jQuery callback that waits for document to load before calling all the separate functions to load the various components of the UI. This function also calls initialize DBs to ensure databases are loaded in before use by the various components. Prof.js

## *NARINE.JS*

**isUser(user)**

Checks whether the username entered by the user exists. If the username is valid, this function will return true, otherwise it returns false.

**getUserPass(currentUser)**

Returns the password on file for the currentUser

**setFocus(name)**

Sets the focus to the input box whose id is passed to this function using the parameter 'name'. If an input field is filled our incorrectly, this function is called to highlight and set the focus on the wrong fields of the form.

**fullName()**

Checks the validity of the input box for user's full name. The user will get an error message on the right of the box when the input it left blank or it contains less than 6 characters. If correct, a check mark will display to verify validity.

**phoneNumber()**

Validates the phone number entered by the user. If the user enters a non-numeric character or leaves this field blank, they will get an error message. If correct, a check mark will display to verify validity.

**verifyNewPass()**

This function takes the input for password verification and compares it to the first input of the new password. If the two passwords don't match or if the second password space is empty, it will warn the user. If correct, a check mark will display to verify validity.

**email()**

Takes the input of the email field and compares it to a specific variable that represents the email pattern (/^.+@.+[.].{2,}$/i). This ensures that the spacing and characters of the entered input match those of an email.

**zip()**

Has the same functionality as the phoneNumber() function to make sure that only numeric characters are entered.

**logInUser()**

Is called when the user clicks LogIn inside Login.html. This function then verifies that the entered username is a valid one and calls logInPass()

**logInPass()**

Called from logInUser() to verify that the password entered corresponds to the password of the currentUser in our 'database' – user.js file. If the password does not match the current username, the function will warn the user, otherwise the page will be redirected to the main dashboard, and the user will be logged in.

**sendIt()**

Verifies that there are no errors after all the validation function calls have been made and lets the new user submit the information.

**sendLogIn()**

Verifies that there are no errors after the username and password have been validated and lets the user login to their GarageShare dashboard.

## *PROF.JS*

**getUserPic(currUserId)**

Sets up user profile information using the current User Id passed to it. Gets the name, profile picture, reputation and friends list for the user and displays them in the appropriate place on the left sidebar.

## *SHIM.JS*

**initializeMap()**

This function initializes the Google Maps API, to allow viewing the search items on an interactive map. It sets up the current user with a marker showing their home on the map and centers the map around this location.

**loadDialog(entryStr, modeStr)**

Populates and opens Item Details Dialog. This allows users to see all the details available for an item, including

Notes and a link to a similar item online. EntryStr is the entry id, modeStr is the mode of dialog, suggests where the item was when clicked.

**loadMiniDialog(title, string)**
This dialog just shows a simple message and allows cancellation. Used to implement message concerning future features. Title is the header of the dialog, string is the message shown.

**loadAlerts()**
Sets up and loads the alert components which shows all the available alerts for the current user.

**clearMapMarkers()**
Clears all the previous map markers from the map.

**displayResultMap(result)**
Displays the map with all the current results loaded. Zooms the map out to show all the available results to avoid user needing to manually search for markers.

**displayAllItems()**
Displays All Items in database, only used for debug

**displayResultList(result)**
Displays list version of search results, each list item opens a modal window with full details on click. Summary details include item name, quick description, owner's name, distance of owner from the current user.

**searchClick()**
Click Handler for search button, searches the database, builds results of items, owners and distances, called displayResultMap and displayResultList on the search results.

**loadNewItems()**
Populates New Items component

**displayBorrowedLentList(result, type)**
Populate borrowed and lent components. Since the same function is used for both components, *type* variable defines which kind of component it is, *result* component contains the results of the database search.

**numDaysTo()**
calculates number of days to Date. Used by displayBorrowedLentList() to generate human readable messages.

**loadBorrowedItems()**
Gets borrowed items results, including owner information, calls displayBorrowedLentList() to display them.

**loadLentItems()**
Gets lent item results, including owner information, calls displayBorrowedLentList() to display them.

# LESSONS LEARNED

The pervasive strength of the 121 course design was the fact that everyone, regardless of initial skill level, appeared to wind up gaining something from the experience. There were several factors that contributed to

this, but perhaps the most potent was the opportunity (and indeed, the necessity) to design and code collaboratively, managing deadlines and challenges in far different ways from standard group projects. There were unique obstacles that arose from this new paradigm, and others that were inherent to the process of design itself. They'll be reviewed in descending order of positivity (and perhaps consequently, ascending order of impact to project).

## The Good

First and foremost, the team felt accomplished for producing a functional prototype in the time allowed by the hasty development schedule, and especially proud of the quality and number of features that appeared in the final product. This was due to factors that included the advanced software architecture experience of members of our team, our overall team cohesion, and intelligent choices early on in the design process, such as feature compromises and choice of development tools.

When exploring options for syncing and managing code across development environments, Dropbox presented itself as a solution that fell on the right side of the line between complexity and convenience, as all of us were familiar with its use. While a "real" VCS like GitHub would have been better suited to managing simultaneous development, we felt retroactively justified in our choice by bearing witness to the teams that suffered drawbacks from the burden of learning Git in addition to the many other technologies essential for development. While Dropbox was not without its own headaches - such as occasional and unintentional "edit wars" between developers - they were easily overcome, and on more than one occasion its basic version control features proved sufficient to recover from otherwise damaging situations such as deleted files.

Although also laden with tradeoffs, an early decision to standardize on the popular Bootstrap web framework made easier the task of building a stable, grid-based layout that could withstand several layers of CSS by many authors without breaking in unexpected ways. Using this solid foundation, the team was able to split large tasks such as building dashboard widgets, knowing the likelihood of a single errant change drastically altering the layout was low. While at times this necessitated "countering" Bootstrap CSS with our own declarations, we felt that overall, the framework enabled a decrease in CSS coding time.

Another early decision that perhaps had the most drastic impact on the finished product was to eschew any kind of SQL backend in favor of TaffyDB, which enables database functionality via JSON objects and facilitated the "moment in time" aspect of our demo. By obviating the need to hard-code the inventory-driven aspects of the site while at the same time sidestepping the pitfalls of running an SQL service, we found a best-of-both-worlds solution that enabled for quicker and more reliable development.

## The Bad

Besides our successes, there were a number of obstacles that hindered our progress - one of which being simply that collaborative design is difficult, and consistent vision sometimes takes a backseat to efficient code delivery. While our initial research and planning adequately set the stage for feature development, aesthetics were not a significant part of our preparatory process; subsequently, much placeholder style became progressively "baked" into the layout, as far-flung modal dialogs and in-Javascript HTML captured the current style at time of writing, and past a certain point, style updates became increasingly inefficient propositions. For future projects, it would seem wise to involve mockups and design revisions at earlier, rather than at later stages.

As mentioned previously, the collaborative aspect of web design was new to some of us, and overall the most impactful part of the course. Considerations included structuring the code to facilitate multiple editors, informal

code checkouts via email, frequent communication about project status, efficient task delegation, regular meetings, and of course, compromise. These did not all occur naturally or early, but were eventually critical to have learned for the overall success of our project.

One example of the importance of communication regards the JavaScript powering the site - were each developer to write code in isolation, there would likely be conflicts in the global namespace (inadvertent similarities in variable names) that could break the project unexpectedly after a manual merging of code. This was of special importance to Narine and Shimona, who wrote the vast majority of the site's client-side scripting, and our overall success likely benefited greatly from their understanding of those concerns.

Another concern present in any development scenario is when to abandon an implementation attempt in favor of a simpler solution. This became relevant during the integration of FB Connect into our login page; after it proved more complicated than the documentation otherwise suggested it should be, the team wisely stopped spending resources on it rather than endangering the project by focusing disproportionally on the problem, and we wound up suffering not at all in its absence.

### The Ugly

Two obstacles were especially difficult to transcend, regardless of team logistics. The first is described simply and colloquially as the "works on my machine" problem: unless multiple computers are available to and frequently utilized by a developer, it can be difficult to identify a rendering quirk of a particular browser version or development environment until the team is physically together, or a teammate is paying particular attention to that area of that project - at which point, design has likely been layered on top of the bug, and identifying the precise code to change can prove challenging.

And finally, CSS itself provided obstacles at nearly every turn. While the team collectively had some prior HTML/CSS skill, styling a webapp such as GarageShare (and doing so collaboratively) was a degree of complexity higher than our average experience prepared us for. Bootstrap greatly eased certain aspects of development, such as the columnar layout and form/button creation, but the tradeoff of a framework showed itself during the deeper styling sessions, which often involved reconciling (and negating) framework-default CSS with new stylesheet declarations - a task which at times seemed to take longer than creating the framework-equivalent feature by hand.

CSS can also demonstrate an unexpectedly high learning curve from the perspective of someone acquainted with modern programming techniques. While many languages offer themselves to similar conceptual spaces and vocabularies, CSS requires both an intuitive feel for the DOM that one is unlikely to have developed previously, and prior knowledge of non-intuitively named and organically-designed selectors with various levels of support and behavioral differences between browsers.

Ultimately, however, our team prevailed, though perhaps without as much visual polish as we had hoped for originally.

## Future Work

Although we were able to complete most of our prioritized goals for the scope of this class, there were many features that we would like to add to our interface in the future. The addition that stands out the most is

of course the implementation of an actual backend for GarageShare. A functional server-side backend, with page generation and database management, would enable us to build a stronger site with refined search options and a more secure database system.

We were unable in the prototype to allow more complex searches such as sliders to control distance of item, selection of categories of items, relevance computations on search results, etc. These features would help the user locate the desired item with greater ease and accuracy.

An important update to GarageShare would be more security for user account information. Currently, there is no implemented session information, password security, etc. With all the files being held on our local machine, it was not possible to implement security that is based off browser-server interactions. With more time on our hands we would also continue working on the Facebook integration aspect of our site and will try to overcome the challenges that were presented by the Facebook Open Graph API.

In the future we would also like to further redesign the GarageShare interface. Throughout the development and during the last weeks of project completion we had great redesign ideas for different color schemes and layouts. However, with the lack of time we decided minimize design changes during the final days before the project demonstration.

# CONCLUSION

With GarageShare, we aimed to create a front-end for a service website that would help users with “sharing stuff they own with friends they trust”. We took our original concept for such a service and gathered some useful data to help drive the design in a user-focused way. From our findings, there appeared to be many aspects of the sharing practice that could be aided with technology to ensure an enjoyable and useful experience for all involved. We addressed the issues by providing specific tools and components to help alleviate the difficulties users had with the borrowing and lending practice. While there is much work to be done toward developing a back-end and more features, we believe that our functional, clean yet interactive solution is a solid first step toward such a service.