Villages Final Report

Client: Circle77 (Jennifer Feighny)
Team: Ben Makuh, Chad McKenna, Jerel Miller, Josh Dinges, Thomas Mesquita

Introduction

Many modern parents struggle to organize the logistics of finding caregivers for their children on a short-term basis. Rather than hiring a babysitter, another option is to "trade" babysitting time with neighbors and create a sort of social economy. Finding these caregivers, however, is only half the battle; individual families need to ensure that everyone who has contributed to caring for their children is rewarded appropriately. The problem is that many families find it difficult to keep track of who took care of whom, how long they've done it, and how often. Social bookkeeping can get complicated quickly! Families need a centralized, organized, secure, simple, and efficient way to manage not only who does what and when, but also other relevant information about the children and the households that they will take care of.

Introducing Villages. The goal of this project was to create a web application that solves exactly those issues. The application has the ability to keep track of caregivers you trust while maintaining accountability via a credit system for the exchange of babysitting services. Throughout the entire application, security and safety are paramount concerns. Villages is built on modern web technologies including Ruby on Rails, HTML5/CSS3, Ajax, and a relational database.

Requirements

1. Functional
    a. Village network system
    b. Pages for households, requests, and search for neighbors
    c. Credit System
    d. Automatic email notifications
    e. Users are registered as part of a household
    f. User verification
    g. Requests for childcare are easily viewable and sortable
2. Non-Functional
    a. Strong security
    b. Database/Ruby on Rails support
    c. Consistent design
3. Use Cases
    a. brand new user visits the application
    See **Figure A** in appendix for an example of what they see.

Precondition: none
Postcondition: User is informed about the purpose and security of the application

Primary Flow:

i. User visits home page in their browser
ii. User watches brief introductory video
iii. Four large links are displayed below video:
    1. Learn more about how it works
    2. Learn more about the vision and purpose of Villages
    3. Learn more about our security
    4. Sign up
iv. User follows links until satisfied

Alternate Flow:

i. User connects to home page
ii. User watches video
iii. User leaves

b. User wants to sign up

Precondition: none
Postcondition: User is brought to household creation page

Primary Flow:

i. User connects to home page, finds and clicks on Sign Up link
ii. Create Account page is displayed, user enters information
iii. Page displays a message about email with verification link being sent to user
iv. User clicks on link from email and is brought to household creation page

Alternate Flow:

iii. b. User does not receive email

c. User wants to find a caregiver to add to their village
Figure B in appendix shows an example of the interface when a user searches for neighbors.

Precondition: User is actively registered for the system
Postcondition: User finds a caregiver

Primary Flow:

i. User logs in
ii. User is at home page
iii. User selects the “Search” box in the corner
iv. The user types in the name of a potential caregiver in the network
v. The user makes a request to the searched-for user to be a potential caregiver
vi. There is a validation email sent to the requested caregiver
vii. The user then re-confirms the caregiver after they have accepted the request

d. User wants to find a caregiver to take care of their children

Precondition: User has a current set of caregivers
Postcondition: User finds a caregiver to watch their children

Primary Flow:
i. User is at home page
ii. User goes to their “My Caregivers” page
iii. Finds a set of caregivers that they want to take care of their children
iv. Makes a request to the caregiver
v. The caregiver is notified
vi. The caregiver can either accept or decline the request
vii. The user is notified of the caregivers response

Design

1. Security

This application is going to have a lot of personal information stored within it, so security is of utmost importance while being cognizant of the trade-off between security and usability.

1.A Sign Up:

Users will go through an initial sign up process to register and use the site. Unregistered users will not be able to access any part of the site other than static informational pages and the sign up form. Once a user is registered he/she must be part of a household before being allowed to use any features of the application. There are two options to choose from regarding households: the user can either create his/her own household or make a request to join an existing household.

The second approach requires the user to search for the desired household. Once found he/she must answer a question or complete a task (ex. calling the household) that the household defines. Once the user submits their answer the household owner is sent an email with the users’ response and then that household is able to either allow the user into the household or deny them access.

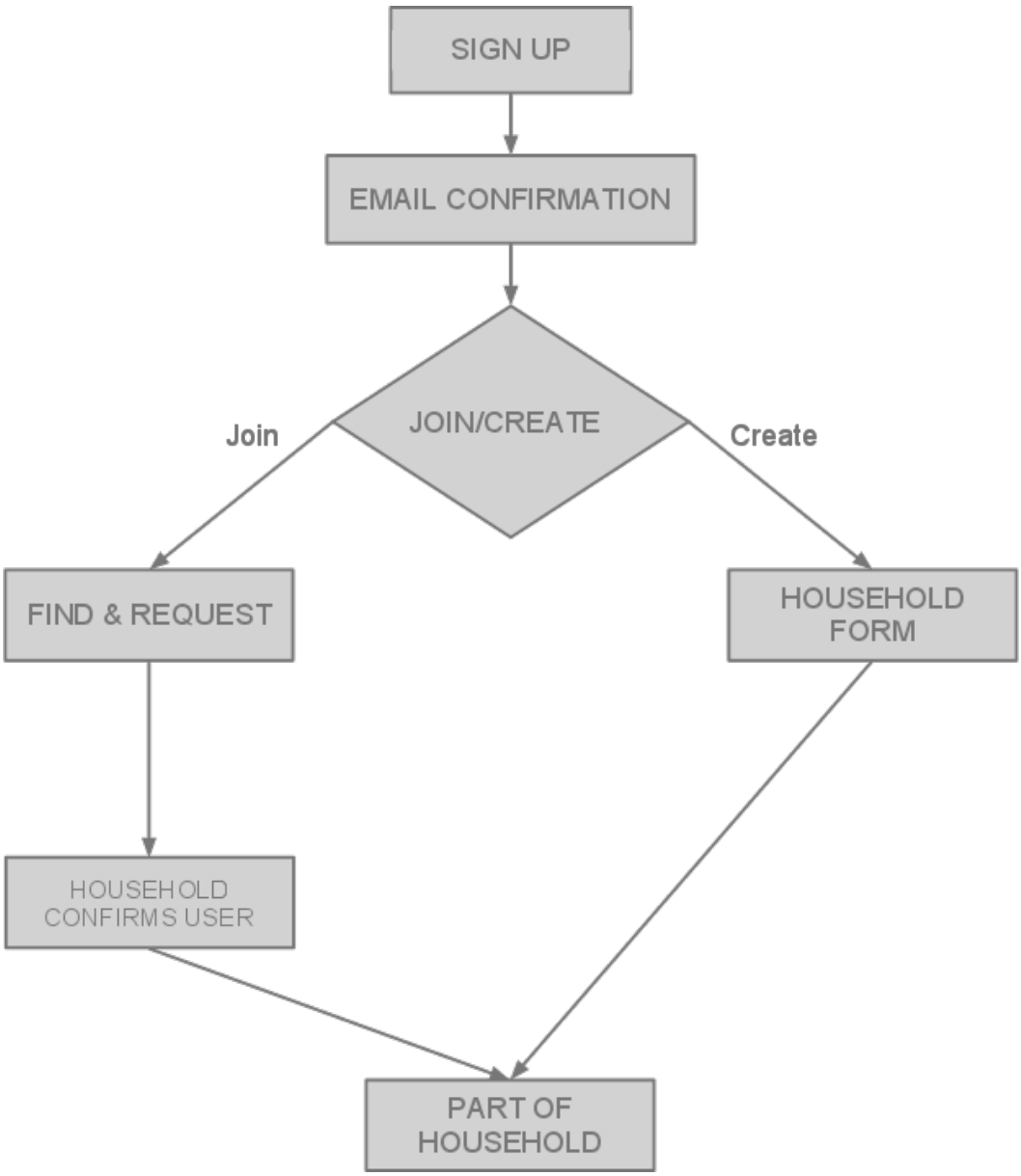

Figure 1. The security process for a user joining or creating a household.

1.B Add Neighbor:

If you wish to add a neighbor inside Villages, the process follows this simple example. Say that we have two Households- the Johnsons and the Smiths- that are not neighbors and therefore are not in the same village. If the Johnsons want to add the Smiths as a neighbor they will simply use the search box in Villages. The Johnsons can search by either the name of the household, the names/email addresses of the caregivers within the household, or a phone number associated with the household.

Once the Johnsons find the Smiths in the search results they will send a neighbor request to the Smiths that explains to them that the Johnsons want to be their neighbor. This request will show the Smiths some basic information about the Johnsons so they

can confirm their identity. The Smiths then have the opportunity to confirm or deny the neighbor request. If the Smiths confirm the Johnsons as a neighbor, each are added to each others' village. At any time during the neighbor request process the Johnsons can cancel the request, and the Smiths can deny the request, thus terminating the entire process.

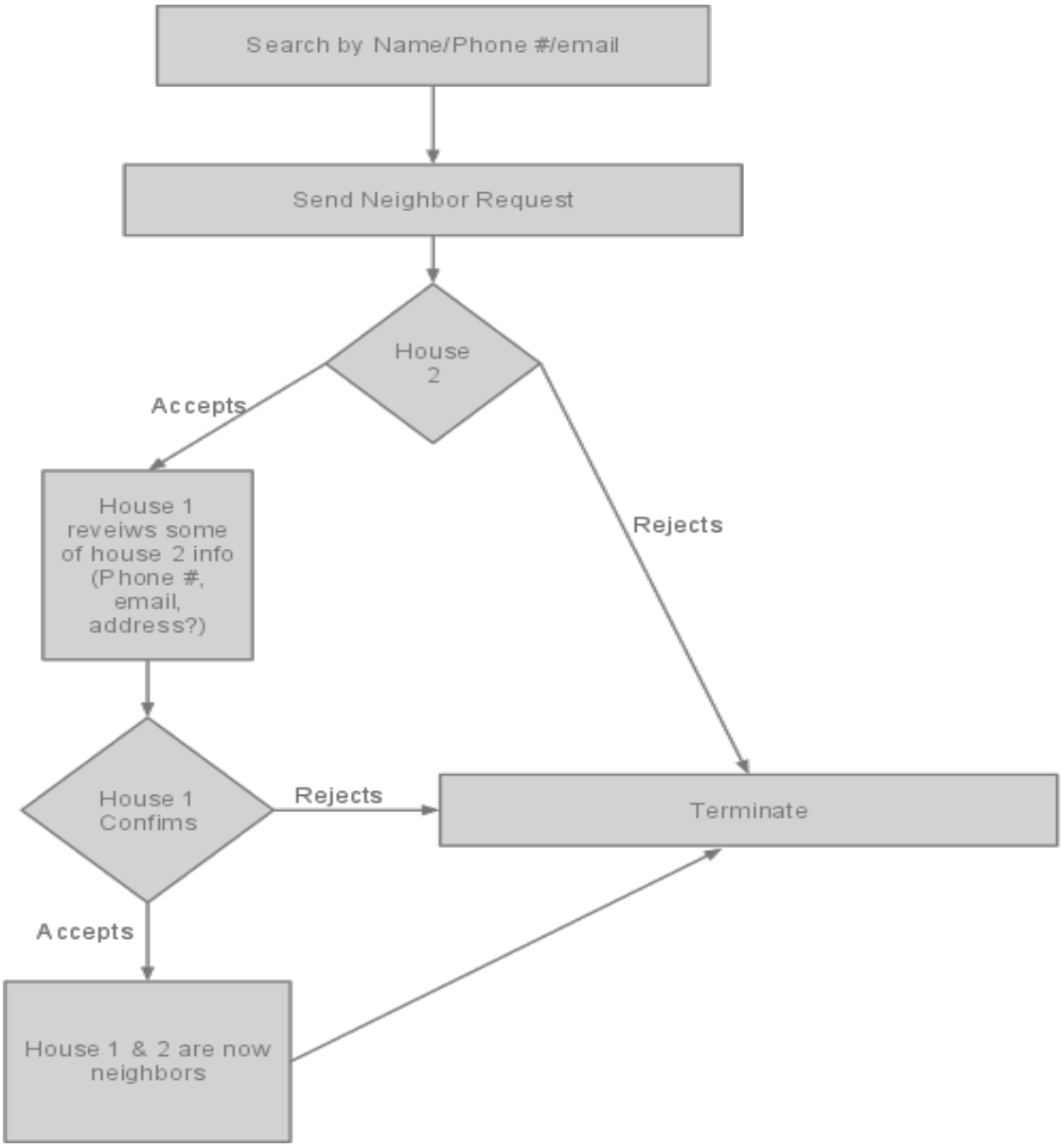

Figure 2. The security process for how a household adds another as a neighbor.

1.C Care Requests:

For a simple example, say we have two houses: the Johnsons need someone to watch their kids and the Smiths are looking to earn a few credits by watching someone else's kids. Furthermore, both the Johnsons and the Smiths are already connected as neighbors. The Johnsons submit a care request to have their kids watched at a specific

date and time. The care request is then broadcast to the whole village. The Smiths see the care request and volunteers to watch the Johnsons' kids.

The system sends a verification to the Johnsons saying that the Smiths have volunteered to fill the Johnsons' care request and the Johnsons can either confirm or deny the Smiths to watch their kids. If the Johnsons confirm the Smiths, then the Smiths are committed to watching the Johnsons' kids at that given time. The Smiths are sent a notification that they have been confirmed to watch the Johnsons' kids at the requested time. If there were multiple volunteers to fill the Johnsons' care request and they were not chosen because it was granted to the Smiths, then all other volunteers will be notified that they were not confirmed as a volunteer for the request.

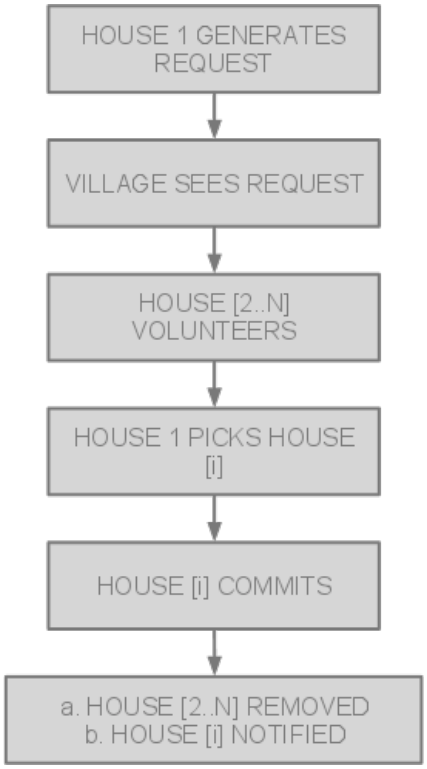

Figure 3. The security process of sending out a care request to your village.

2. Database Schema:

Our database schema has seven main entities that make up the backbone of our application. The main entity that all the other models revolve around is the Household model. Child and Caregiver both model the *individuals* that belong to a Household, whereas EmergencyContact and Address are complex *attributes* of Household. Request models each and every care request that Households create.

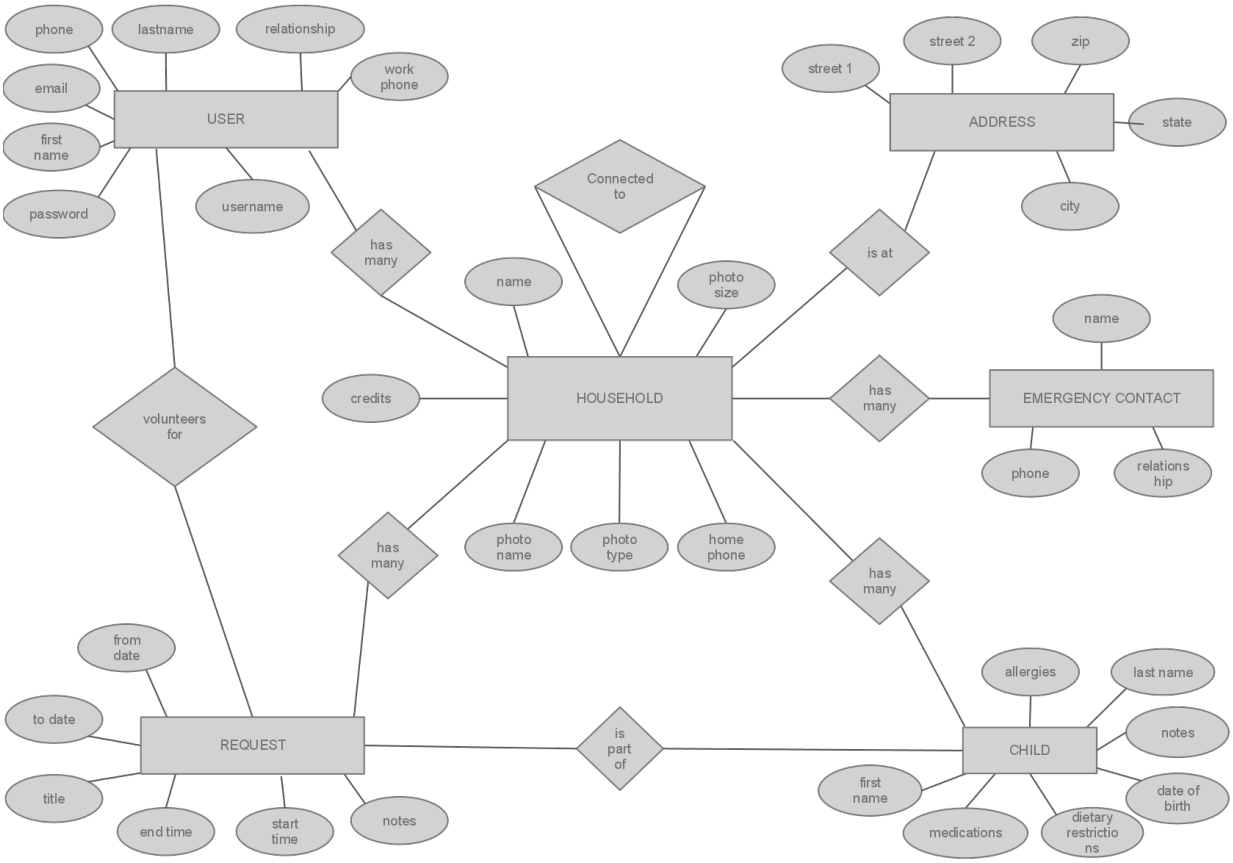

Figure 4. The database schema, represented as an Entity-Relationship Diagram.

3. Pages:

This application will contain the following pages with different levels of authorization:

a. Unregistered User
   i. Splash screen
   ii. How it works
   iii. Our Vision
   iv. Our Security
   v. Terms of Service
   vi. Create an Account/Sign-up

b. Registered User without Household
   i. Create Household
   ii. Search for Households to join

c. Registered User within Household
   i. Search for Households to add to one's village
   ii. Create Request
   iii. Household management
   iv. Edit settings
   v. View other Household requests (within village)

Implementation Details and Results

Tools:

For the development of this application we used Ruby on the Rails Framework. We chose to use Ruby on Rails because of its highly streamlined design that allowed us to create an agile programming environment as a team. Ruby itself also has an extensive set of Gems that enabled an incredible amount of functionality with relatively few headaches.

We used Git as a code repository, Heroku for our staging and production servers, and various Internet browsers for testing compatibility. One major change that we had from our initial design was that we used a traditional relational database instead of MongoDB because of support issues. Rails natively supports relational databases through ActiveRecord, whereas to use Mongo we would have had to use a gem which is not as easy to set up on Heroku (which can be a challenge as it is).

Results

The final product was made available to the client at http://villages.heroku.com. This will function nicely as a beta testing site, as she plans to run a small test within her own local neighborhood and network of friends. A few changes to the codebase and the design will need to be made before scaling larger, but that is outside the scope of our project.

Scope and Project Progression

This project was written from the ground up with no prior models or examples to work from. While the application at least somewhat resembles a contemporary social network such as facebook, it is different enough to the point that we had to stray far from the ordinary user model and implement a custom, combination user/household model. In the common social network the user is the functional unit upon which all security and validation rides. Villages departs from that model by using the household as the functional unit. Users log *into* the household and then interact *as* the household with other neighbors. The household can also have many users, which is again quite a significant change from the common user model.

The danger of our approach is that it creates a steeper learning curve for a new user. To lessen the difficulty of learning a new system, we took great care in making the application as streamlined and intuitive as possible. Still, that does not completely eliminate all of the problems. Because of this, the sign up process is designed to be an informative walk through the different pieces of one's household in hopes that once signed up, the only thing left to learn is how to interact with neighbors and how to use the interface.

Conclusions and Future Work

The project has been completed up to an informal 1.0 status defined as a working, stable application that was feature-complete except for a few items which we were not allowed to touch because of field session rules. What it does *not* include is financial handling, true identity and

location verification, internationalization, or a mobile-specific interface. This means that the application on delivery to the client will *not* be ready for final production. It will be very usable for beta testing purposes, and it will be ready for future software engineers to complete the aforementioned 2.0 specs.

Advertising space, initially a 1.0 spec, was moved to 2.0 since at 1.0 the site will not be ready for production. Since creating the space for advertising only involves some simple CSS repositioning, it made sense to leave the interface as clean as possible so as not to confuse beta testers.

As a team, we were able to learn a great deal of knowledge regarding modern web application development. In addition, we got to experience a very agile software development environment in which our code base was growing and changing every day. We started with absolutely no code and completed a functioning and relatively complex web application.

Glossary

ActiveRecord - a database Object-Relational Mapper that abstracts the details of storing data away from the programmer.

Gem - a module for the Ruby language

Household - A network of caregivers under a physical household that may be responsible for taking care of the child.

MongoDB – A schema-less, scalable, high-performance, open source, document-oriented database that does not use the relational model or SQL.

ORM – An object-relational mapping that abstracts the work of interacting with a database away from the programmer.

SQL - Structured Query Language

Village - Based of the idea “It takes a village to raise a child.” Used to refer to the network of households

Appendix

Figure A. The splash screen of the application, which visitors first see when they come.

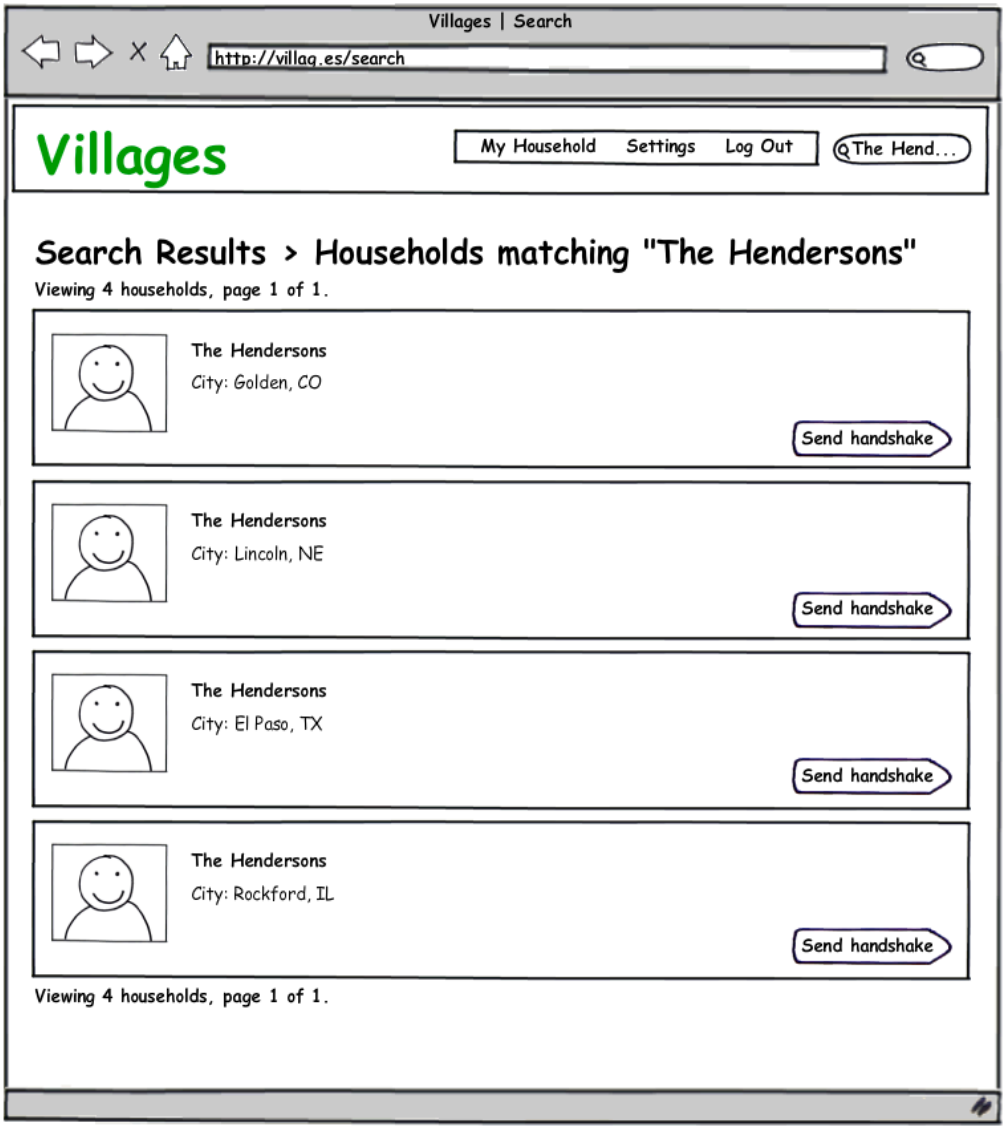

Figure B. A user searches for neighbors.