lendr

Design | 11.25.14

Ryan Frankel, Dennis Garcia, Alex Romero, Aaron Suarez

Overview

lendr is an application that will allow users to find each other easily to lend and borrow items.

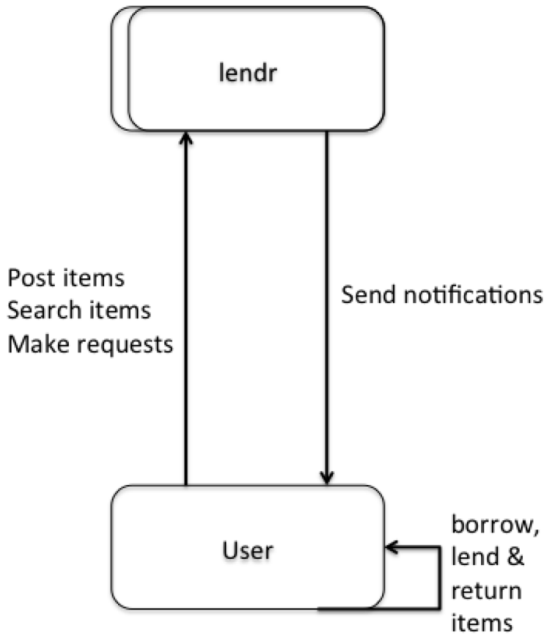
Motivation

In the past month alone, individuals across 4 living groups have made at least 200 requests to borrow items. Many of these requests go unnoticed or ignored as they get lost in the spam of living group email lists. What if a person in need of an item never even needed to send this request in the first place?

lendr has two main purposes:

1. encourage people to lend their unused items
   - we want to motivate people to share what they have with others
2. facilitate finding and borrowing items in nearby locations
   - we want to make it easy for the users to find an item they need and borrow it from someone that is nearby, instead of buying it when it's not necessary or have to send mass emails to find someone that is willing to lend it.

Context Diagram



Design Model

Concepts

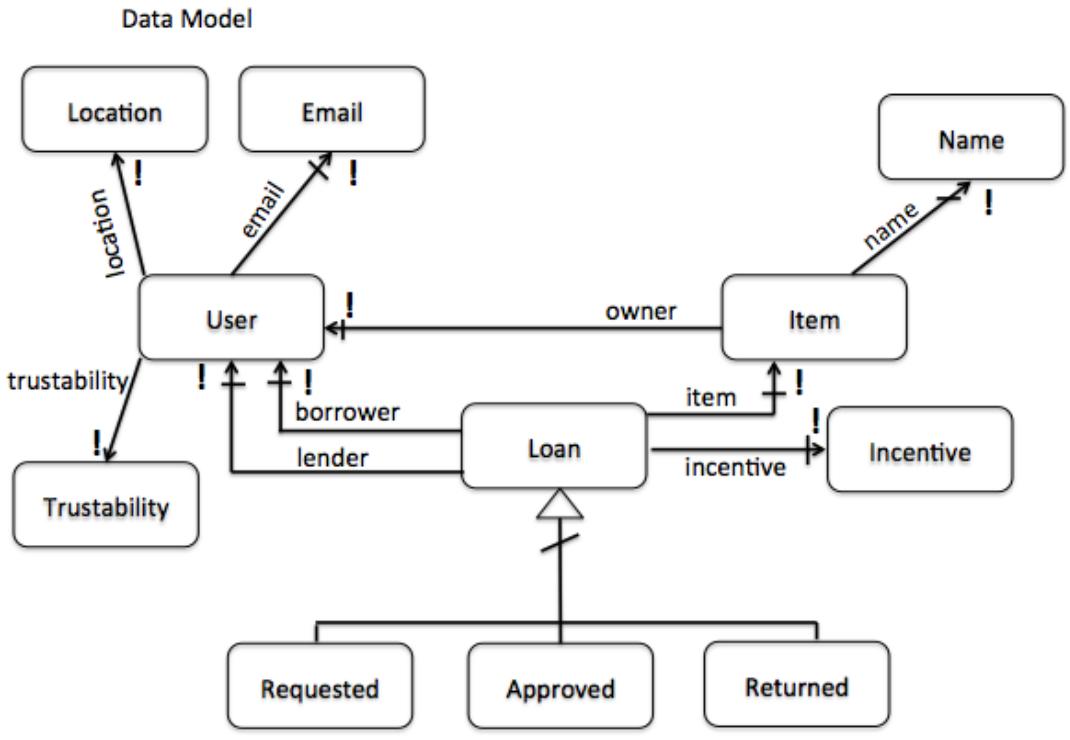
Loan → Agreement between lender and borrower where lender gives an item to the borrower in exchange for an optional incentive and for a limited amount of time (motivated by both purposes).

Trustability → Rating of users by other users that takes into account the punctuality of return, quality of the items, friendliness, overall satisfaction, etc so that both lender and borrower feel more comfortable with the **loan** (motivated by both purposes).

Incentive → Offer made by the borrower to the lender when requesting to borrow an item (motivated by purpose of encouraging lenders).

Location → Place of residence of the users that allows borrowers find items that are physically close to them (motivated by the purpose of making it easier to find items in nearby locations).

Data Model

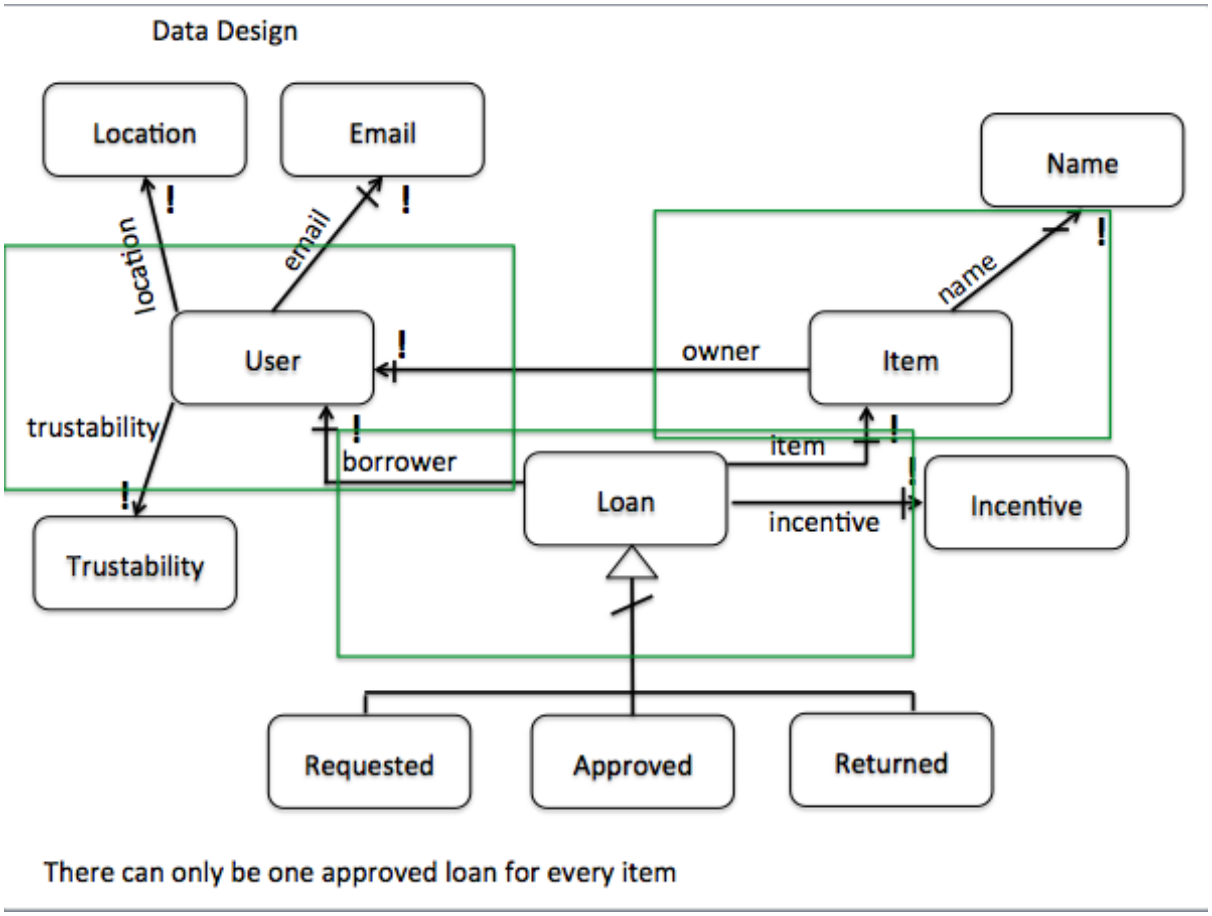


There can only be one approved loan for every item

There can only be one approved loan per item and one returned loan per approved loan
Trustability will be a rating from 0-5 determined by the loan rating of borrowers and lenders
Changes(+) and justifications(-):

+ Terms changed to incentive (what the borrower offers the lender)
    - Only important term we care about (for now at least)
+ Subclasses of loan (requested, approved, returned)
    - Keeps track of the state of a loan
    - Cleaner than separate objects
    - Easier to handle than booleans in a single object

Data Design



User also has name and password
Loan also has a temporary mutable cached rating from borrower to lender (lender to borrower is automatic) and is only required for returned loans

Behavior

Security Concerns

Security Policy

- Every user has the ability to request to borrow items from the website and to initiate a return request for a specific item that he/she already borrowed.
- Every user has the ability to post items to lend out to the website.
- Every user has the ability to give trustability to other users based off of their opinions of the transaction made with said users.
- Only the owner of an item has the privilege to delete an item they own
- Only the owner of an item has the privilege to edit the descriptions of any of the items he/she owns.
- Only the owner of a specific item has the ability to approve/deny a request from another user to borrow that item and also to approve/deny a request from another user to return that item.
- Only the owner of a set of items is allowed to see the full list of items they own.

Threat Model

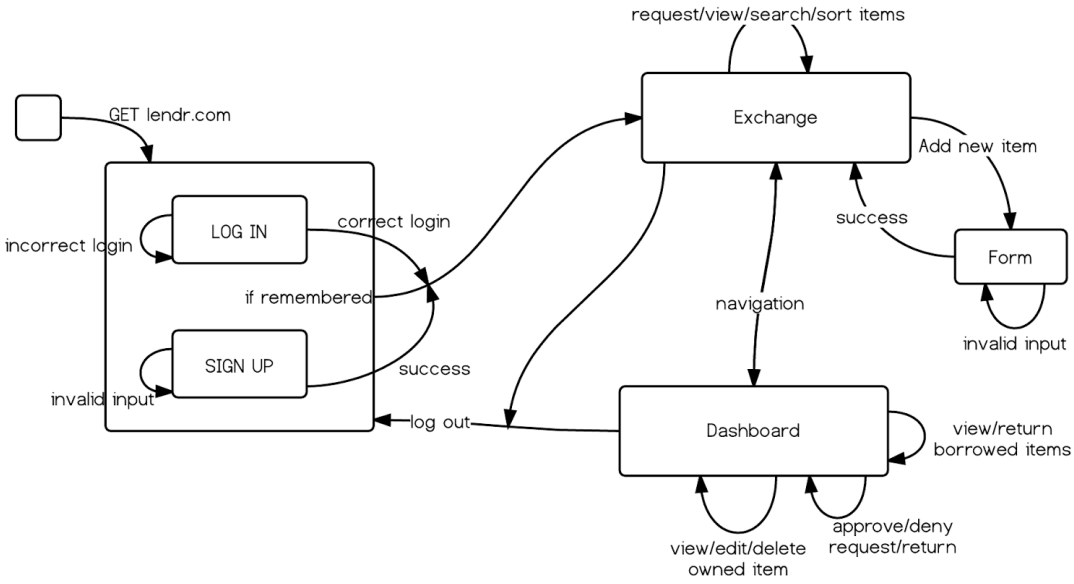
- A user with basic credentials can construct requests or make requests via forms.
- Unauthenticated user can pose as a normal user and post, and borrow items without appropriate permissions.
- User with basic credentials can construct a request to see all of the items that belong to a specific user and also maybe gleam any other specific information about a certain user (such as living group).
- A user can construct a request to give him or herself a higher trustability or tamper with other users' trustabilities.
- A user with basic credentials may try to act on requests on behalf of another user
- We assume that attackers do not have access to our physical servers
- We assume that attackers cannot guess other users passwords
- We assume that attackers cannot create false MIT email addresses

Mitigations

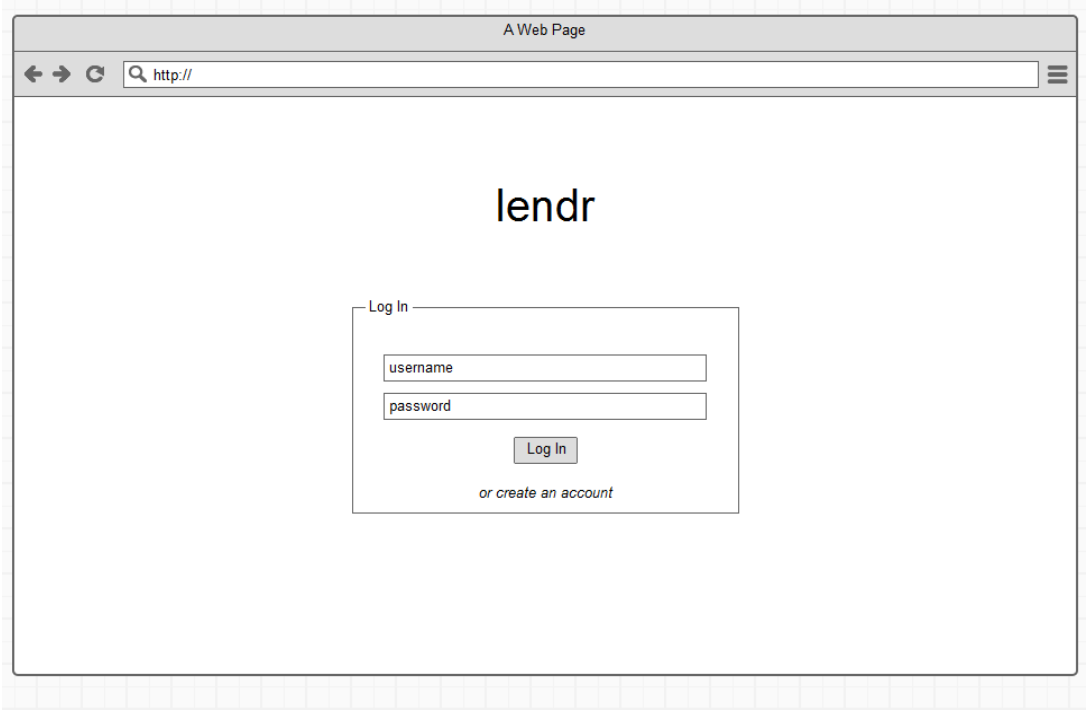
- To ensure authentication, we implemented a login system with unique @mit.edu emails and private passwords. This ensures that users are who they say they are.
- To ensure authorization across all users (logged in or not), we will restrict all API methods to logged in users as well as access to views of the app.
- To protect against injection attacks through form inputs, we will stringify and clean the submitted inputs before making use of them. This will help us guard against XSS attacks as well.
- To protect against higher level attacks such as CSRF, we will use both built in middleware provided by Express as well as Helmet.js -- middleware that aids in securing express apps.

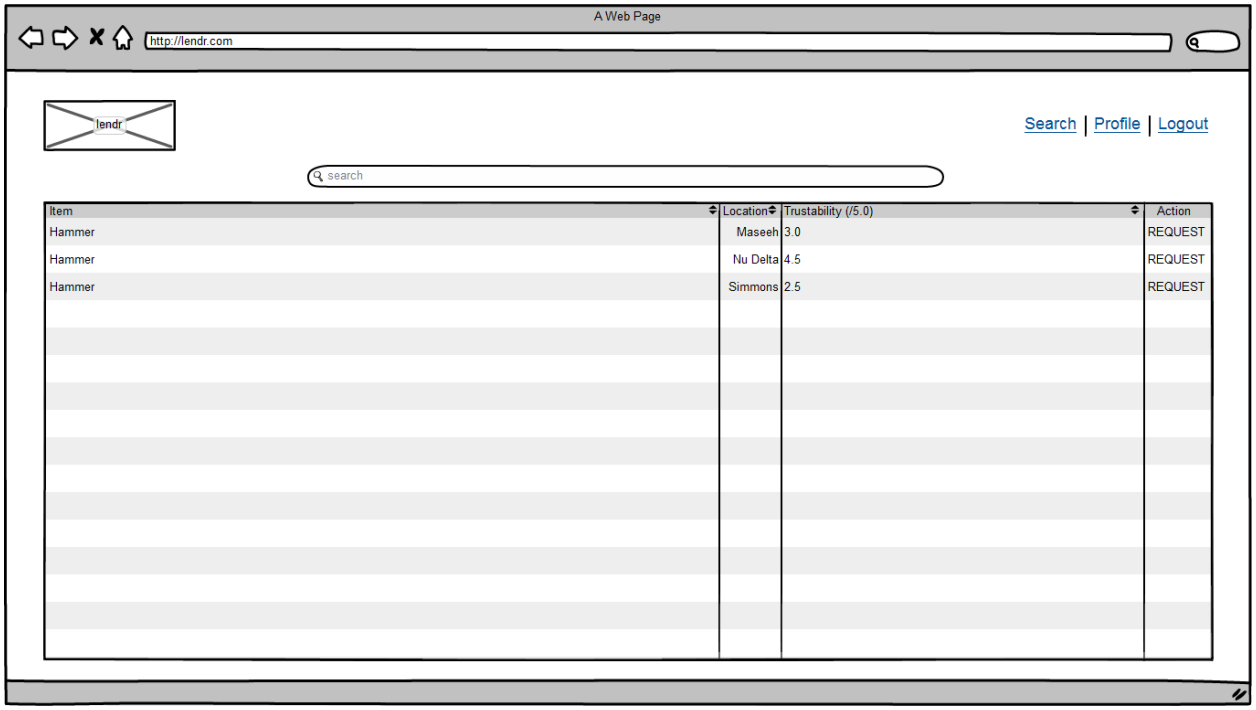
User Interface

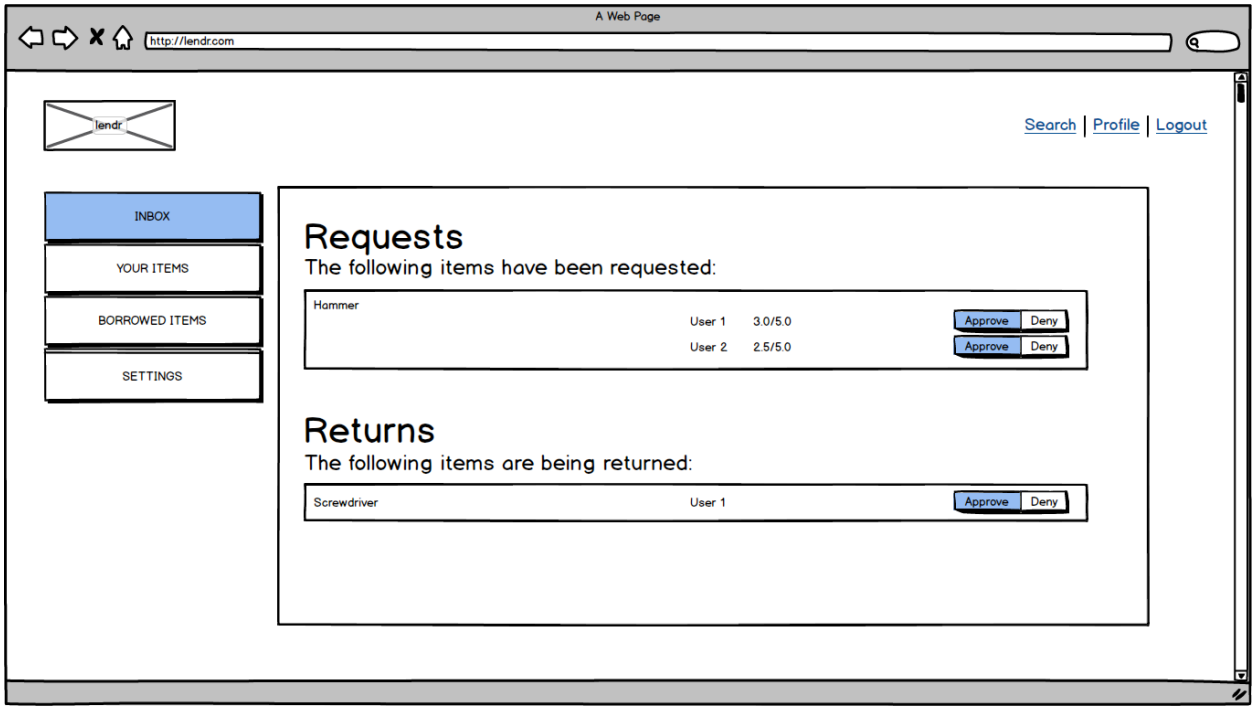
App Flow

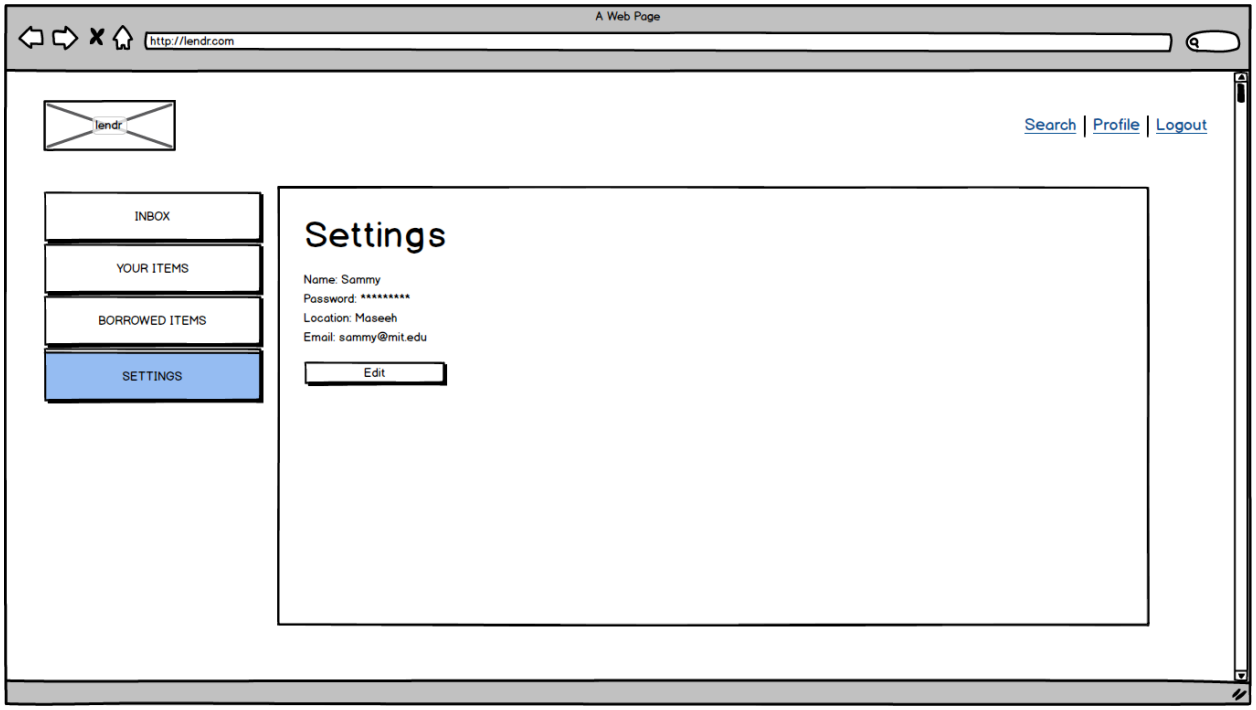


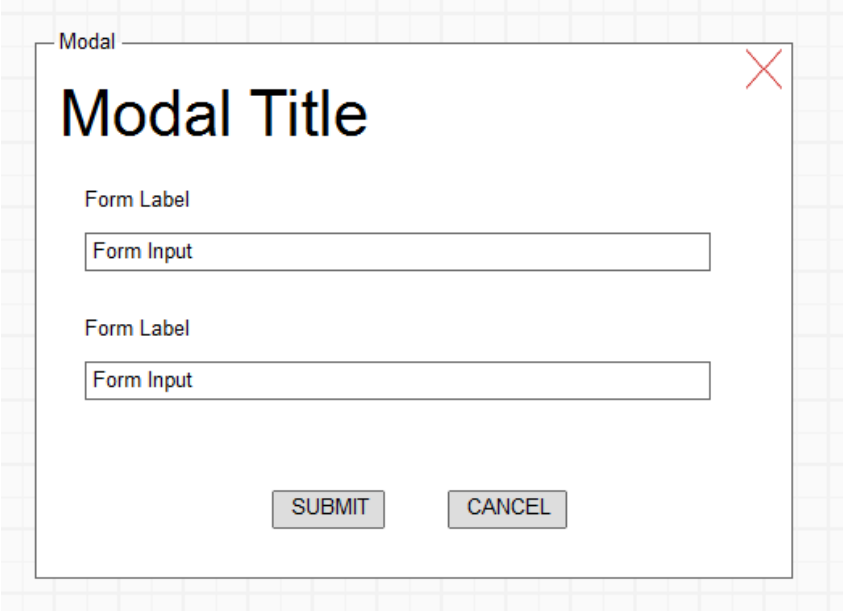
Wireframe




A Web Page
http://lendr.com
lendr
Search | Profile | Logout
search
Item
Location
Trustability (/5.0)
Action
Hammer
Maseeh
3.0
REQUEST
Hammer
Nu Delta
4.5
REQUEST
Hammer
Simmons
2.5
REQUEST


A Web Page
http://lendr.com
lendr
Search | Profile | Logout
INBOX
YOUR ITEMS
BORROWED ITEMS
SETTINGS
Requests
The following items have been requested:
Hammer
User 1
3.0/5.0
Approve
Deny
User 2
2.5/5.0
Approve
Deny
Returns
The following items are being returned:
Screwdriver
User 1
Approve
Deny


A Web Page
http://lendr.com
lendr
Search | Profile | Logout
INBOX
YOUR ITEMS
BORROWED ITEMS
SETTINGS
Settings
Name: Sammy
Password: **********
Location: Maseeh
Email: sammy@mit.edu
Edit


Modal
Modal Title
Form Label
Form Input
Form Label
Form Input
SUBMIT
CANCEL

Challenges

What is the benefit for lending out items?

Challenge: Giving people a reason to want to lend items

Options:

1. Lender posts item with monetary price:
    + Allows lenders to set prices and potentially make money
    - Less interactive, fun, and creative
    - Potentially expensive for borrowers
    - Makes app similar to a CraigsList for lending
2. Borrower makes a request with an incentive (chosen):
    + Still allows lenders to get rewards for lending items
    + Allows for fun, creative, and potentially delicious incentives
    + Allows borrowers to offer only what they are willing to give
    - The borrower may not know what the lender is interested in receiving

Is the location of an item a concept?

Challenge: Determining whether or not location/living group should be a concept. From the beginning, we had intended to allow users to search for items by living group or location, but we were unsure whether it qualified as a concept.

Options:

1. No:

    Explanation: One could argue that location is simply an attribute of an item or user. It is also not considered to be a unique and new concept.
2. Yes (chosen):

    Explanation: Location is essential to one of our purposes - to allow users to more easily find nearby items that they need to borrow. Therefore, it is a concept.

Is the item a concept or is there some other concept that is more essential to our application?

Challenge: To capture the main purpose of our app - borrowing and lending items - within a concept.

Options:

1. Item as a concept:
    + Users post, lend, and borrow items in our app
    - Items are not a unique concept to our app, and items
    - Does not capture what is being done with the items
2. Loan as a concept (chosen):
    + Captures the essence of our app
    + Essential to our core purpose
    + Encompasses the idea of an item

How should items and users point to one another?
Challenge: Within our data model, we were initially unsure whether users should point at items or vice versa.
Options:

1. Users point to items:
    + Makes sense intuitively, because every item belongs to a user
    - Every operation on an item would require inefficiently iterating through all users and their list of items
2. Items point to users (chosen):
    + Improve efficiency of find operations by searching on items (which occurs more often than searching on users)

How should transactions be represented in our data model?
Challenge: How to represent a transaction in our data model.
Options:

1. Each item points to a borrower:
    - Additional booleans needed to tell if an item currently has a borrower
    - Would lead to a series of inefficient nested find operations
2. Each user points to a list of borrowed items:
    - Items and users would have relations to each other--a bad practice
    - Would lead to a series of inefficient nested find operations
3. Create a "loan" object (chosen):
    + Points to a borrower and an item (which points to the item's owner)
    + Leads to the simplest possible data model
    + Allows for three subtypes: requested, approved, and returned
    + Reduces the number of booleans needed to keep track of states of items
    + More efficient implementation

What is the best way to implement search?
Challenge: How to implement a search over all item descriptions
Options:

1. Use the entire string input and see if it is contained within any of the item descriptions
    + Easy implementation
    - Requires exact matching of string input and item description
2. Use an external search library (chosen)
    + Allows for a more fuzzy search that will return all possibly related results
    - Not designed by us, so behavior is not customized or controlled
    - Returns items that may be unrelated (can be too fuzzy)
    - More difficult to implement

How should we implement the get method for lent items?
Challenge: Finding an efficient way to query over Loan and Item schemas in order to get all lent items.
Options:

1. Iterate over all loans and find loans with an item of the user
    - Very inefficient and potentially very costly to query over all loans
2. Iterate through all items that belong to the user and find which items are included in loans
    + More efficient to only query over the items of user
    + Narrows down the search significantly in the first query

How to know if you already requested a return?
Challenge: How to keep track of the returns of the items you've borrowed for button display
Options:

1. When getting your borrowed items, get returns and determine whether one has been made for each item.
    + Avoids changing data models/schemas and/or routes
    - Asynchronous calls give problems, nesting the borrowed items.
2. Change route to return information needed (chosen)
    + Avoids changing data models/schemas
    + Easier implementation
    - API methods may return ambiguous information

How to keep track of session?
Challenge: What information do we store in req.sessions?
Options:

1. Store the entire user object
    + Would give us access to all fields would need without querying the database
    - Gave us a maximum flow stack error
2. Store the user id (chosen)
    + Avoids any errors
    + Still allows us access to all information we need
    - We have to query the database to get this information
3. Store the info we need individually in req.sessions (i.e. req.sessions.location)
    + Easy to access info we need
    + Avoids errors
    - A lot of things to keep track of