Kathryn Siegel, Sylvan Tsai, Ari Vogel
6.170 Project 3.1: Minimum Viable Product
October 27, 2013

Minimum Viable Product

primary author: Sylvan

Overview

- Other methods of splitting a bill require:
    - Calculations done by cell phone calculators or by hand to find the appropriate amounts
    - Remembering to bring cash to a group outing and accounting for people who forget to bring cash
    - Being able to break large bills and find change
    - Verbal I-owe-you's that will doubtlessly be forgotten
- Split.li:
    - Removes the need for cash, change, and calculations
    - Allows you to charge your friends via Venmo before even leaving the restaurant
    - Leaves a record of who-owes-who by text and email, and on both parties Venmo accounts

Identification of MVP

- **Split.li**, an app that allows friends to evenly split a bill at a restaurant
- Simply input your friends' names onto our receipt along with the total amount and tip. Split.li will do all the math by dividing the costs evenly and sending your friends Venmo charges

Key goals and purpose

- Facilitates group outings by removing the hassle of calculating who-pays-what and who-owes-who
- Offers a convenient way of reimbursing someone in a timely manner

Motivations for development

- Provide an avenue for easily splitting a bill at a restaurant without having to do all the math and factoring in tax/tip by yourself. A night shouldn't be ruined by trivial calculations on cell phone calculators and searching for appropriate amounts of cash and change.

Features Subset

- **Venmo integration.** Sign-up and log-in done through Venmo, allowing prompt reimbursement through Venmo charges and payments.
- **Bill calculator.** Remove the hassle of adding purchases, dividing tax, and scrambling for tip dollars: Split.li will do the math for you by splitting everything evenly.
- **More reliable reimbursement.** Split.li sends Venmo charges immediately after a receipt is entered, encouraging friends to send payment before even leaving the

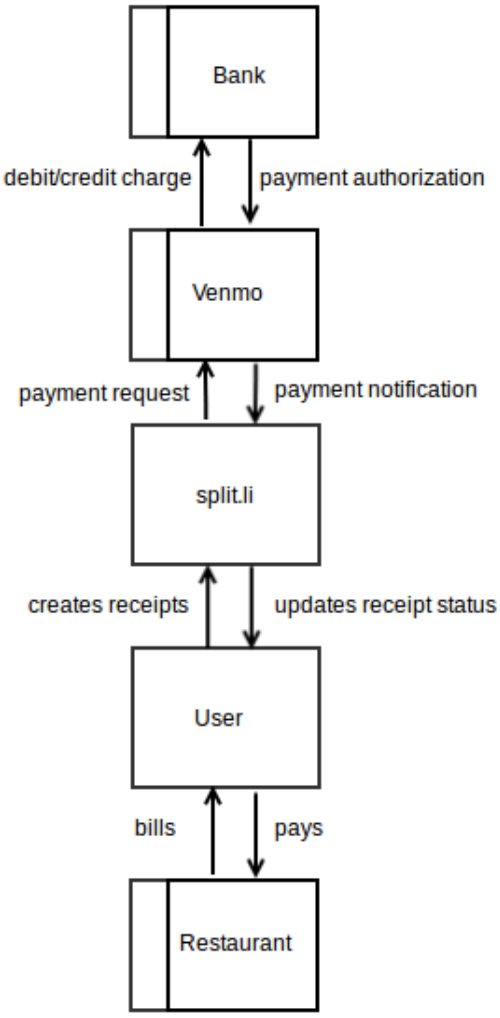
restaurant.

Postponed Features & Issues

- **Mobile-friendly.** Use Split.li anytime at your convenience. (Not MVP but began implementation during Phase 2)
- **Proportional bill split.** Pay for only what you bought. In addition to evenly dividing the total cost, Split.li offers purchase aggregation by itemizing the bill, allowing for a bill split (including tax, tip) proportional to the amount each person spent
- **Splitting item cost.** Account for multiple people sharing an item (e.g. two people split an appetizer) in how much each person should pay.
- **Progress bar.** Track how many people have reimbursed you for a specific receipt thus far.
- Postponed implementation of user profile, contact page, session expiration

Context Diagram

primary author: all



Concepts

primary author: all

Key concepts

- Split.li facilitates peer to peer interaction with Venmo, taking a receipt with a total cost, tax, and tip, and sending Venmo charges on behalf of our User.
- Split.li does purchase aggregation by splitting the entire bill, including tax and tip, evenly among a group.

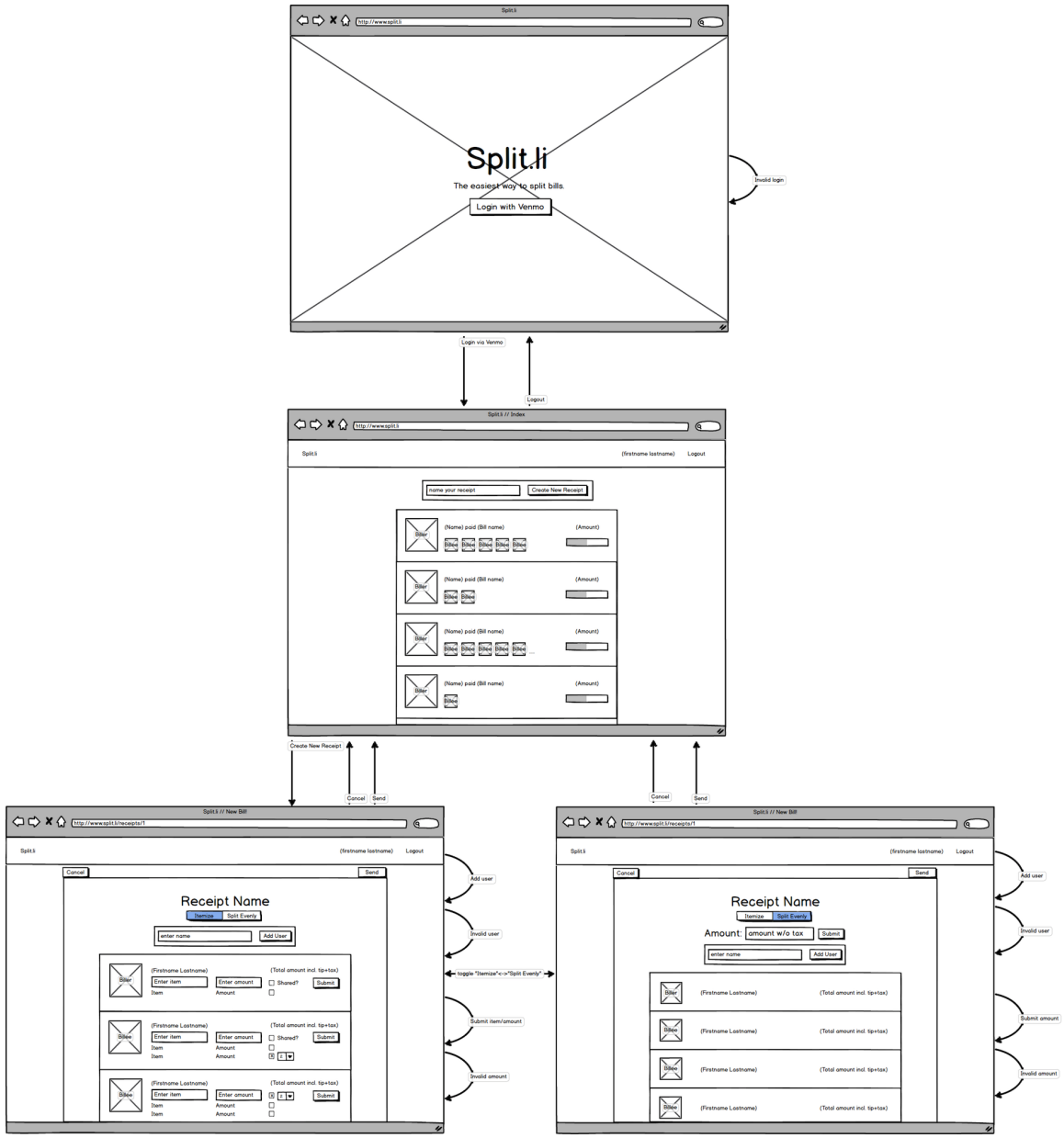
- Split.li requires a User to sign in with their Venmo account. That User can create a Receipt and add Purchases to the Receipt.
- A Receipt contains a total amount (subtotal cost + tax), a percentage for tip, and Purchases.
- Purchases represent purchase records. For our MVP, they include a name and a U.S. phone number, through which Venmo can send charges based upon the Receipt's total amount and tip.

Data Model

- Textual constraints
    - A Venmo charge cannot be sent unless the Receipt contains a total amount that is greater than zero and more than one Purchase.
    - A User cannot create a receipt without a name; i.e. a User must input a non-empty string as the name of the receipt.
    - When adding purchases (i.e. a purchase record with a friend and a number), a name and a valid phone number must be inputted. The name must be a non-empty string, and the phone number must be a ten-digit number of any commonly-accepted phone number format.
    - A receipt must have a non-zero and non-empty amount.
    - A receipt's tip must be a valid number; otherwise, the tip defaults to zero.

User Interface

primary author: Katie



(to view full-size, click here: http://web.mit.edu/ksiegel/www/mockup_6170.png)

Security Concerns

primary author: Ari

- Split.li makes secure payments with Venmo, utilizing their API's OAuth.
- Split.li does not expose OAuth tokens to prevent fradulent Venmo charges
- Split.li's use of Venmo for all payment concerns ensures that securing OAuth tokens will secure users payment permissions, as we do not keep any balances between users.
- Split.li does not store any sensitive information, such as passwords, which can be exploited if made available
- Split.li does not use insecure third party plug-ins
- Split.li does keep a history of past group purchases for use on the newsfeed page. However, a person can only view a past purchase if they are both a Split.li user and involved in the purchase, either as the person who paid the entire bill or as one of the people who paid that person back.
- Split.li enforces SSL for additional security in verifying the site and encrypts the traffic between client and server.

Design Challenges

primary author: Sylvan

- How to add purchase records dynamically to a receipt
    - Use forms to create purchases
        - Pros: implementation is much simpler using forms
        - Cons: not user-friendly. The user will likely input total amount and tip first, but upon adding a purchase, those fields will be cleared.
    - Use Ajax to save purchases
        - Pros: more user-friendly and intuitive to use since total amount and tip fields will not cleared unexpectedly
        - Cons: implementation with Ajax is not necessary for the minimum viable product
    - Our MVP solution was to use forms in order to get the simplest thing that works, with the intention of implementing Ajax to save purchases for the final product. (Note: since we had time, we did implement Ajax saving but we do not consider it part of our MVP)
- How to add other users (purchase records) to a receipt
    - Do this through the friends list of a user (fetched from Venmo)
        - Pros: most people don't know their friends' phone numbers. It is much more convenient for the user to use information already saved in their Venmo profile

        - Cons: many people do not friend everyone they know on Venmo since it is not a social networking website. Instead, they only add people as the need arises, which means they would not be able to add their friends through Split.li

    - Input phone numbers manually
        - Pros: this is easier for us to implement since we can use our current skillset in validating phone numbers and place responsibility with the user to get the appropriate phone numbers instead of using Venmo's API. We ensure that the user does not run into the problem of not being Venmo-friends with someone they are splitting a bill with.

        - Cons: this would be irritating for users who are Venmo friends with people they are splitting the bill with, especially if they split the bill with the same person often since Split.li does not offer a way to save past purchase records to reuse a name and phone number.

    - For the MVP, we opted to input phone numbers manually, since it is the simplest thing that works and thus more appropriate for this phase of the project. In next phase we want to use the user's Venmo friend list in order to make our app more user-friendly, with the option to also input phone numbers manually in the case where the user is not friends with someone they are charging.

- How to create architecture for splitting evenly that will carry over for itemized purchases in Project 3.3
    - Create a friend model: since each input for the splitting evenly is a friend with a name and phone number, each friend will have a purchase - an amount to pay - associated with them. Then for itemized purchases, a purchase can be associated with a friend
        - Pros: Easier to understand the idea of a 'friend' or other person to split the bill with, rather than simply a list of purchases. Easier to understand the association between a purchase, friend, and receipt if friend is an actual model.
        - Cons: Easy to have multiple instances of the same person within the friends table, as it is uniquely created for each receipt creation. This nullifies the power of ActiveRecord, and acts more as a hack than an actual design choice.

    - Simply use purchases as purchase records, as there isn't much extra information being stored there. When we itemize the receipt, the receipt will have a set of associated Venmo friends which will be part of the receipt, and each purchase will have an associated venmo_user_id.
        - Pros: Can recreate the relation of 'friends on a bill' by writing good model code which handles this by simply looking at purchases on a bill, and

looking at the venmo_ids associated with that purchase. Then the ActiveRecord paradigm is maintained and there is no repeated information in the database
        - Cons: Requires more planning ahead with model code. Also, this method can cause confusion as it is not straightforward to think about a relationship without the 'friend' or 'venmo_friend' directly in the schema.

    - We chose the second option for our MVP as it will make expansion of features in our final product simpler. This will enable scaling to be more fluid since there will not be unnecessary entries in a 'venmo_friends' table, and the object-oriented paradigm of ActiveRecord will be maintained. Therefore, the code will be easier to understand and build upon in the future.

- How to handle Venmo's access tokens with security in mind
    - Store them in the database since the access tokens do not expire
        - Pros: Implementation is simple as the access token only needs to be fetched from venmo on the first login for that user. Then on subsequent logins, the same token can be used.
        - Cons: This method causes security concerns as the access token can be used fraudulently if someone can access the database

    - Store the access token in the session, and clear the access token as the session is destroyed
        - Pros: This is the more secure method as the access_token is not stored within our database. Therefore, a database breach would not reveal all our users access_tokens
        - Cons: requires more work since the access token must be parsed and stored each time a user signs in.

    - Ultimately, we chose the second option because we did not want to insecurely store the access tokens in plain text within the database. By storing the access token in the session and then clearing the access token when sessions are destroyed, we do not keep sensitive information on file.

References

- Third-Party Gems
    - twitter-bootswatch-rails, twitter-bootswatch-rails-helpers
    - nokogiri, rest-client
    - therubyracer
    - omniauth-venmo
    - rack-ssl-enforcer