[Project Overview](#)

[Motivation](#)

[Definitions](#)

[Concepts](#)

[Anticipated Misfits](#)

[Data Model](#)

[Security Concerns](#)

[APIs](#)

[User Interface](#)

[Design Risks](#)

[Design Choices](#)

Project Overview

GroceryShip is a web application that facilitates peer grocery delivery. Students from the same college can post requests for groceries they need with a detailed description of the item/s. Students who are doing grocery shopping can then claim and fulfill these requests. Requesters specify how much they're willing to pay as a delivery fee and which pickup/delivery locations (dorms, living groups, other on-campus places) they prefer. Payments and ratings are made once delivery is complete and both parties indicate this in the app. We will focus on an implementation for MIT students first.

Motivation

MIT students are generally busy and do not have time to get groceries in grocery stores which are all quite far away from the main campus. This web application will allow those who are busy to get the groceries in less time and those who make it to grocery stores to earn pocket money by delivering groceries for peers.

Existing solutions include the delivery services of each grocery store (if it exists) and 3rd party delivery services such as Instacart. The main difference is that GroceryShip is a peer to peer delivery service where your peers help you buy your groceries. Users can also specify how

much they're willing to pay as delivery fee whereas other services have a fixed delivery fee. Other services also have a minimum price or number of items you need to buy in order to avail of delivery, whereas with GroceryShip, you can request for any items.

Definitions

1. Shopper: user at a grocery store who is about to complete requests
2. Requester: user who needs items from the grocery store but can't go; makes a request on the app with the list of items he/she needs

Concepts

1. Request (MVP)

- **Purpose.** Allow requesters to inform shoppers about items they need and how much they are willing to pay for the delivery
- **Operational Principle.** A user fills out a request form under Requests, filling out each field to describe each item, the pickup location preference, and the delivery tip he/she is willing to pay. The user clicks "Make Request" and all other users will be able to see the request on the request feeds.

2. Peer Delivery (MVP)

- **Purpose.** Allow shoppers to fulfill requests
- **Operational Principle.** A shopper views all the requests from a feed and clicks claim on a request to deliver items. The requester is notified about this and is asked to contact the shopper to agree on a pickup time. The shopper buys the items, fills out the actual price for each item, then notifies the requester to meet at the pickup point and at the arranged pickup time. The shopper meets the requester and gives him/her the items. Finally, the requester makes the payment and gives a rating for the shopper.

3. Rating

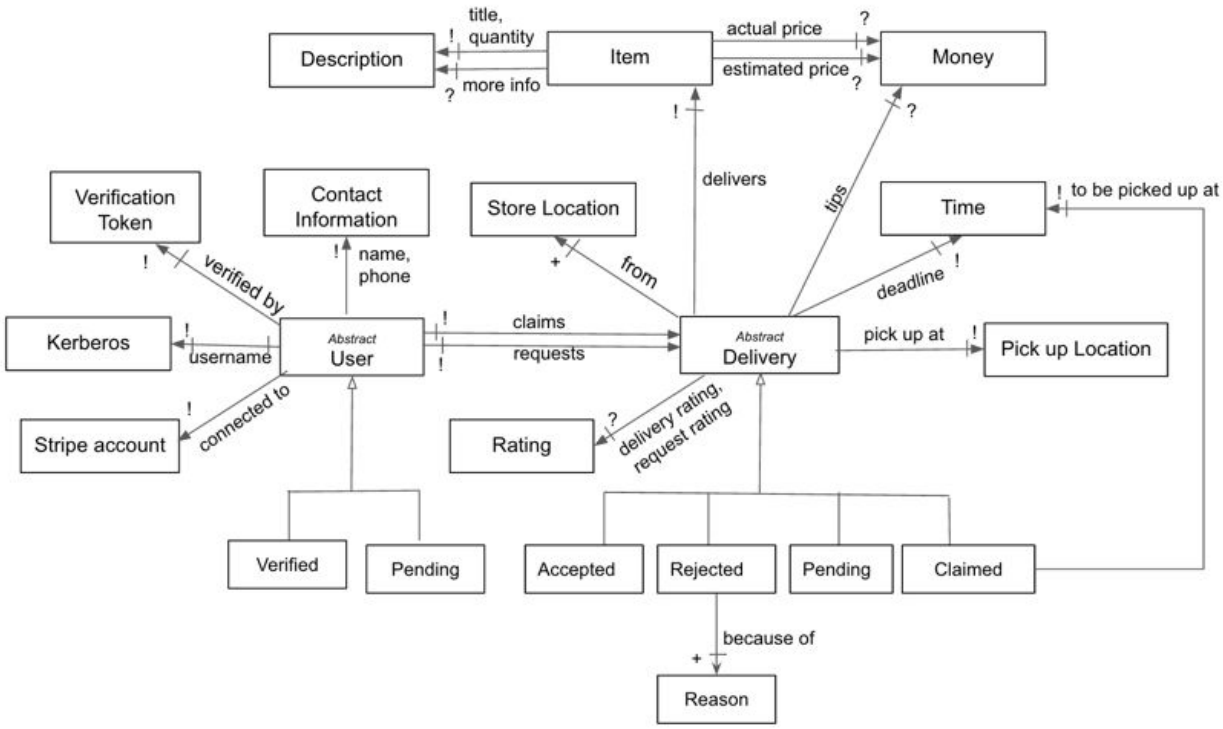
- **Purpose.** Allow the administrator to filter irresponsible users and provide better service quality and to incentivize well-behaved users
- **Operational Principle.** After each delivery, the requester and the shopper give each other a rating by selecting from 1-5 stars, where 5 stars is the best rating. If it is not 5

stars, they will indicate with a comment why it is not. Users with an average of 3 stars or below are suspended.

Anticipated Misfits

- Users might randomly request items and not pick them up.
- Users might randomly claim requests and not buy the item/s and not deliver them.
- Shoppers might buy the wrong item.

Data Model



- Explanations
    - Requests start out as a Delivery that is Pending.
    - Pick up location is set by the User who requests the Delivery.
    - A Verified User means that the User has shown that they own the kerberos they signed up with by clicking on a link in an email sent to that kerberos.
- Textual constraints
    - Requester and shopper of a request must be different users.
    - Delivery pickup time is before the request deadline.

    - The user who gave the delivery rating must have claimed a delivery corresponding to that rating. Similarly, the user who gave the request rating must have requested a delivery corresponding to that rating.
    - A delivery can only have delivery rating and request rating if it is an Accepted or Rejected delivery.
    - Users cannot claim their own requests.
    - Pick up time must be earlier than the due time.
    - Item of a Pending delivery cannot have an actual price.
    - All Users start out as Pending and can change to Verified. Once a User becomes Verified, it stays Verified.
- Insights
    - The concept of request and delivery turn out to be the same because a request eventually turns into a delivery so we decided to model them as one set, Delivery.

Security Concerns

- Summary of key security requirements and how addressed
    - No exposure of database details so that user and payment information aren't exposed and stolen.
    - Secure payments will be done using Stripe, an existing secure payment method, so that none of the transactions will be compromised.
    - Login validation will be used so that users can only make requests for themselves and do actions on them.
    - Passwords will be hashed so that no one's passwords are stolen.
    - Money amounts will be validated in the model (e.g. validate that the delivery tip is not negative) so amounts in transactions are will always be accurate.
    - Login is needed to access the database and all the actions on it are logged so that internal attacks have a threat of punishment.
- How standard web attacks (such as XSS, CSRF, etc) are mitigated
    - XSS: use Handlebars to render the text content of the request posted by the requester
    - CSRF: use secret session token from both cookie and form (double submit)

    - Login CSRF: use anonymous cookies with double submit
- Threat model
    - Attackers can be users who post requests containing text that invoke malicious scripts.
    - Attackers can be from other domains/websites.
    - Outside attackers don't and shouldn't have access to the server. Internal attackers (developers working on GroceryShip) fear the threat of punishment.

APIs

Dependence on APIs is a concern in itself because if any of those services fail, then GroceryShip might fail as well. Thus, we document these dependencies. GroceryShip will depend on the following APIs:

- Stripe

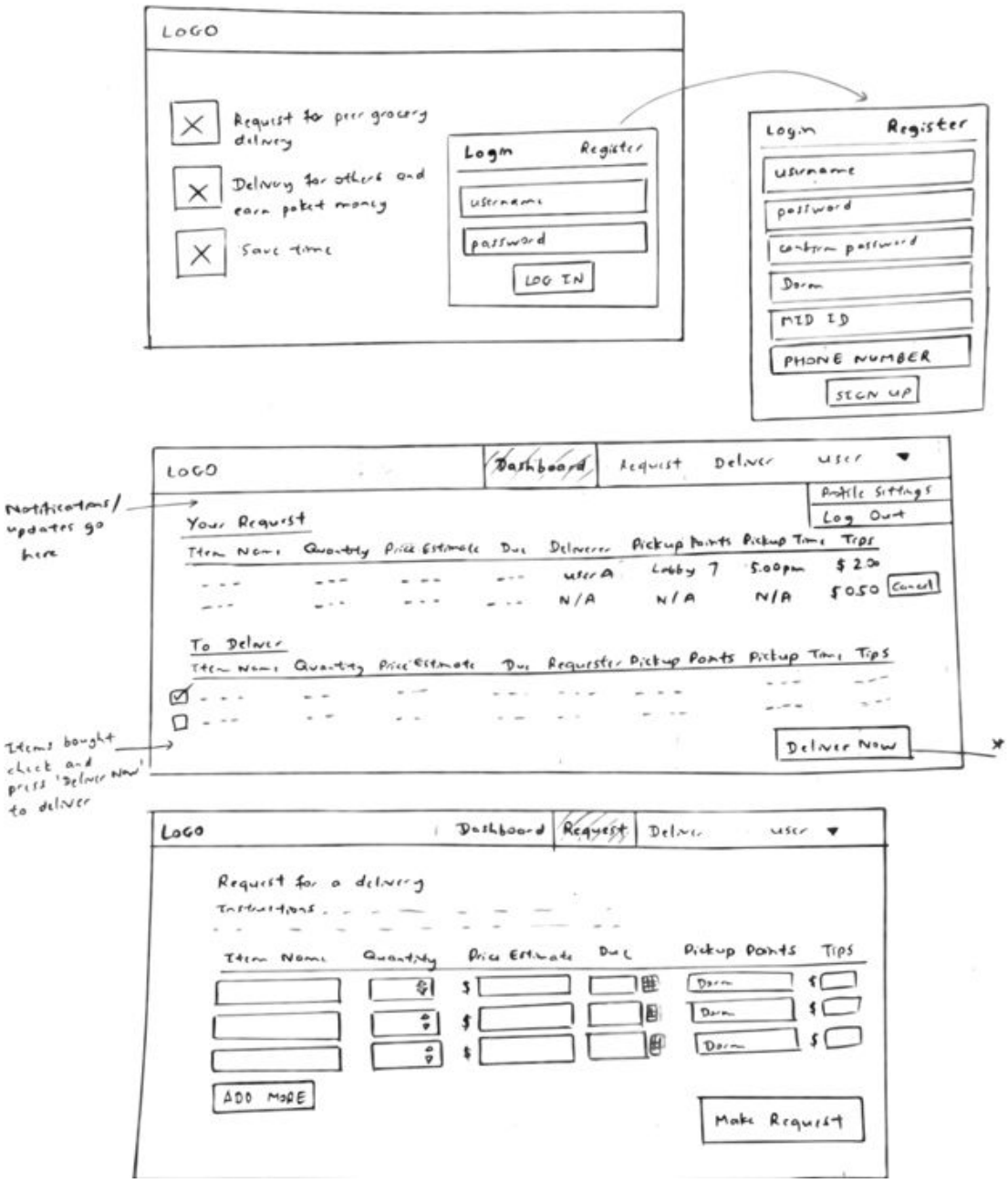
  GroceryShip will use Stripe to process payments and Stripe has an API that can be used with Node.js that we will use. This is how we will use it with the logic in our backend:
    - **On signup,** users will be asked to connect with their Stripe account (and create one if they don't have one yet). Users will be sent to the Stripe authorization URI with the appropriate parameters and callback function once the user logs into Stripe and authorizes access for GroceryShip. The callback function leads the user back to GroceryShip along with the their Stripe account details like the user's Stripe account ID.
    - **To get other Stripe account details**, we will call *stripe.accounts.retrieve()* with the Stripe account ID to get the user's Stripe email. The Stripe account ID and email will be stored as part of the user's GroceryShip account.
    - **For payments,** we will ask the requester to enter his/her credit card details and use Stripe to process the payment. We will set the shopper's Stripe account as the destination for the call to *stripe.charges.create()*. We then store the Stripe charge object ID from the response for possible future reference.

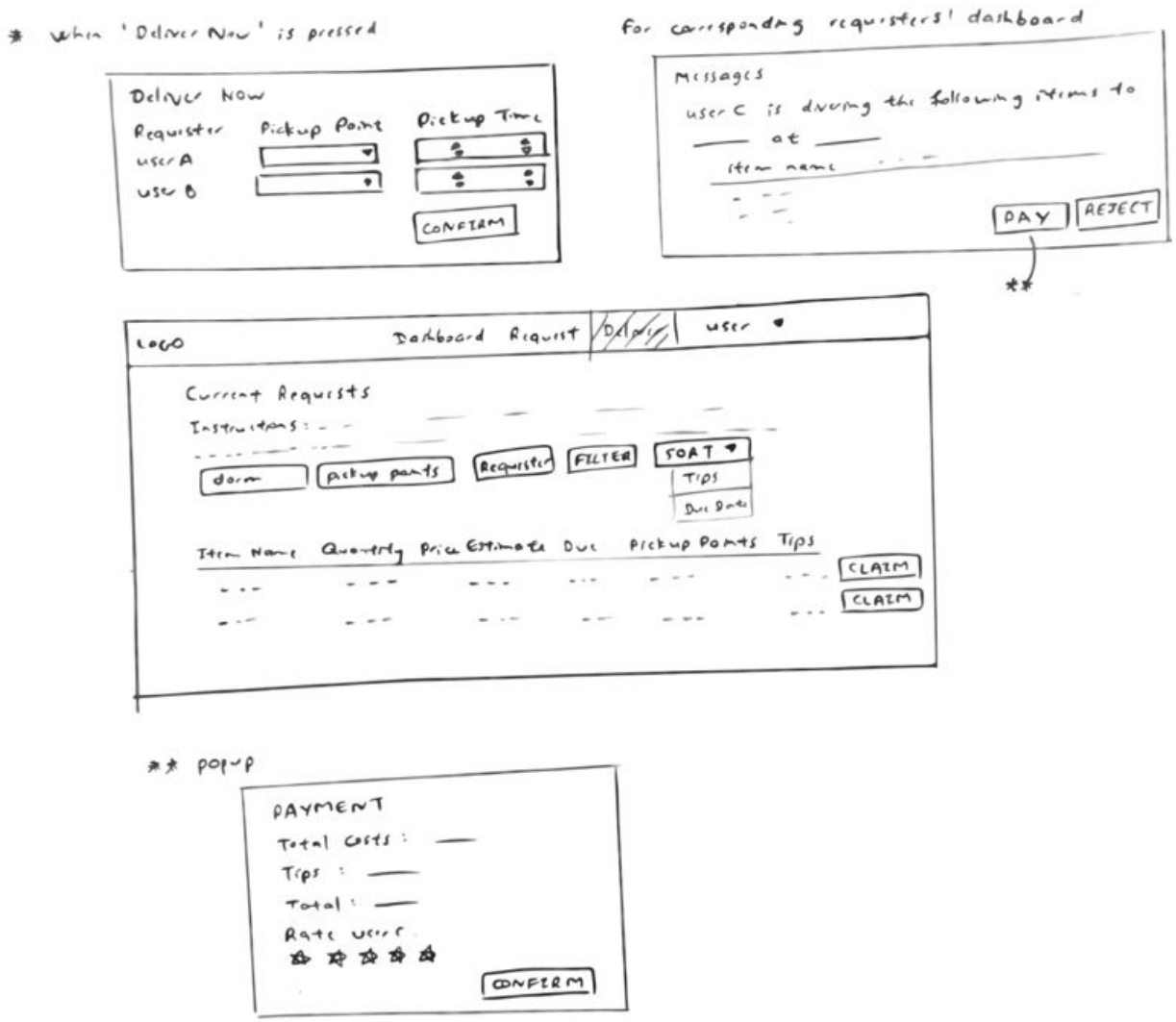
- MIT People

  Since we are limiting GroceryShip to MIT, we want to ensure that the kerberos users input on signup is valid. We also want to make sure that it is not simply an MIT mailing

list because this would defeat the purpose of ratings. To solve this, we query the MIT People API and send an email verification on user signup. We will simply make a GET request to /people/{kerberos} which will give information about the person with that kerberos if it exists. GroceryShip will then take the user's first and last names from the result. If the kerberos doesn't exist, it sends an error code back. Users with an invalid kerberos will not be allowed to sign up.

User Interface





Design Risks

- Deliverers might buy the wrong items.

  Solution:
  - Only allow grocery stores with full-refund policies.
  - Instruct the requesters to provide complete and precise item name and if possible additional description
- Irresponsible requesters or shoppers.

  Solution:

  Use user ratings and let user accept terms and conditions of using the app
- Outsider takes the requested items.

Solution:

- Give the requester and deliverer a shared code.

- Fake kerberos or use of MIT mailing list as kerberos

  Solution:

  - Get access to the MIT People Directory through the MIT People API to verify existence of kerberos and that it is not a mailing list.
  - Send a verification email to that kerberos to verify ownership.

- Fraudulent credit card

  Solution:

  - Use established secure money transaction method such as Stripe.

Design Choices

- **Concepts:**

  Problem: Whether to just have the concept of “Delivery” or both “Request” and “Delivery”

  Option Chosen: We chose to have separate concepts called “Request” and “Peer Delivery” because in reality, they serve different purposes and it is better to have a concept for each purpose.

- **Data models:**

  - Problem: Whether to have “Request” and “Delivery” as separate fields, or merged as a single field.

    Option Chosen: We decided to merge the two concepts into a single field. This is because almost all the fields/information pertaining to “Request” is relevant to “Delivery” as well, as they are both requesting/delivering a common item, at a common pick-up point, etc. Hence, it is more sensible to represent them as a single transaction between the deliverer and the requester.

  - Problem: Our concepts are “Request” and “Peer Delivery”. One of the challenges is that a requester may want to request for multiple items at once and we allow

that. However, there might not a deliverer who can get or find every item on the list especially if the items can only be bought from a specific grocery store.

Option chosen: After a request is made, instead of creating just one request, we will break request into multiple requests each containing only one item. That way, a shopper can claim any item he/she has time to find/get for the requester.

- **User interface:**
  Problem: Whether it is an issue to also include notifications on the dashboard along with the status of requests and deliveries because it might get too cluttered; what are the important things that a user needs to see right away

  Options:
  - Status of requests and deliveries, and notifications **(chosen)**
    - Pro: everything that the user needs is in his/her homepage
    - Con: it might get too cluttered with many requests and notifications
  - Just status of requests and deliveries (use another tab for notifications)
    - Pro: clutter in the dashboard is prevented
    - Con: users won't be able to see the notifications right away