SSH New Feature - Household grocery order

Engineering Design Review
Author: Ben Thompson

Introduction

Rather than visiting a local supermarket or greengrocer, some students now prefer to have groceries delivered to their house, with most supermarkets offering to pick and deliver groceries for an additional cost. This can often be because the nearest supermarket is at least one mile away, which is inconvenient for most students with restricted access to transport. The additional cost can be detrimental to student finances, and also introduces additional carbon expenditure compared to walking. A solution to this is for all participating students within a house to place their groceries order together, only requiring the one delivery fee and trip.

Practically implementing this however can be of great difficulty. One student must take total responsibility for the order, including:

- Keeping track of what groceries each member of the house has ordered.
- Dividing the monetary expense based on the groceries ordered.
- Initially paying for the entire order and then relying on other housemates to reimburse them.

Managing to execute this without error can be most difficult, meaning some housemates may be dissatisfied with their order. To expect a single housemate to bear this responsibility is often why many student households do not consider this approach.

We propose to extend the current SSH infrastructure to bear the responsibility of the single student. There already exists individual profiles within the SSH system for each student, so introducing an individual groceries cart to each profile will not be difficult. The shopping cart of all participating housemates can then be aggregated into a household cart, for which an order can be placed. This software can be integrated into the SSH Console Table and SSH App, meaning it will be accessible to all students within a household.

It would also feature live product costs and promotions according to data supplied by partnered supermarkets.

Since this would be a unique service with no initial UK competition, it offers a unique selling point for the SSH ecosystem, potentially introducing new customers solely for this feature. Existing users would also experience a great increase in the quality of product they receive, as it is a feature accessible and useful to all. Additionally, It would be possible to limit this service to SSH Cloud members, which introduces further revenue/subscription opportunity for the business.

Goals and non-goals

- **Goal:** Introduce shopping baskets for each individual user within the local SSH network, and aggregate all baskets into a combined household basket.
- **Goal:** Summarise the total cost, and break that down into the personal cost for each user.
- **Goal:** Integrate current prices and promotions into application system, in line with those supplied from supermarkets. Make this accessible to users when searching and adding products into basket.
- **Non-goal:** Propose grocery suggestions to users.

Design overview

The new shopping cart feature will exist in a separate tab within the SSH app and SSH Console Table. Within this tab, the main visual element will be a scrolling list containing all of the products within the current basket. Each row within this list will show identifying information for each product, such as the name, along with aggregated information regarding the entire basket.

- A product image provided by the partnering supermarkets.
- The name of the product as given by the supermarket.
- Current promotions associated with the product if applicable.
- Current cost per item.
- Quantity and total cost for individual user.

It is important to allow users the function to search for a product to add to their groceries. A search bar should be included just above the described list. This should open a pop out window using the same scrolling list indexed by the value in the search bar

Personal cost can be displayed alongside the user name associated with the basket. We will include a separate view option for each user to view the group order as a whole.

Handling delivery fees
Ultimately, it will be up to the household users how they choose to split the delivery fee associated with each order. We will implement a majority vote including a proportional split[1] or fixed split[2]. The individual cost is given by the sum below:

$$(1) : \frac{\text{delivery fee}}{\text{users in order}} \qquad (2) : \frac{\text{user order value}}{\text{total order value}} \times \text{delivery fee}$$

There should also be the option for a single user to absorb the entire delivery fee for the instances where that is applicable.

Existing and future data
Presently, we record the personal data of each user of the house, meaning there already exists a user identification value, *user_id*. This means we can create new shopping baskets for each user related to their user identification value.

Table	**Relevant fields**
basket	basket_ id, user_ id, order_ id, last_updated
order	order_ id, date_ created, date_ placed, fee_split
basket_contents	basket_ id, product_ id, quantity

Working through the structure, the first table is *order*. When a new order is started (not placed), a new record is created in this relation, assigning an order id number and recording the date of order creation. Once an order is placed, the *date_placed* field can be updated, and the *fee_split* field also depending on the choice of the housemates.

Each user who chooses to participate in this order is then created a record in the *basket* relation. This records the user and order associated with each shopping basket. An associated *basket_id* is then also created to speed up user basket generation

The *basket_contents* table is where the products contained within each order are recorded. This will take advantage of the database of products provided from the various supermarkets that SSH are partnered with. The *product_id* provided will be used to record the product added to basket, with the *quantity* also recorded.

Building functionality from this data
Shopping baskets can be built by selecting all the *basket_contents* records with a given *basket_id*. Retrieving the required data for each shopping basket can be achieved through the following SQL query.

For a scheduled job, we need to retrieve the list of products and quantities related to a users shopping basket, then retrieve the product image, product name, promotions and price for each of these products. This gives us the data to further calculate the total cost, resulting in all the data required for each row in the shopping list determined above. The query below would produce the data needed for each job, when supplied a *user_id*.

```
SELECT p.image_url, p.name, p.promotion, p.price, bc.quantity
FROM product, basket, basket_contains
INNER JOIN basket_contents AS bc ON basket.basket_id = bc.basket_id
INNER JOIN product AS p ON bc.product_id = p.product_id
WHERE basket.user_id = <user_id>;
```

Furthering this, the household order can be produced by aggregating the baskets with the given *order_id*. This can be achieved by a query similar to the following.

```
SELECT product_id, sum(quantity)
FROM order, basket, basket_contains, product
WHERE order.order_id = basket.order_id
AND basket.basket_id = basket_contents.basket_id
GROUP BY basket_contents.basket_id
```

It is unlikely for the entire order to be accessed frequently, most likely just before placing the order only. So, it make sense for the household order only to be aggregated when the household order page is navigated to and when an order is placed. This can be further optimised by caching the resulting order and timestamp in the SSH Hub. If there are any user baskets with a sooner *basket_updated* timestamp, then it will be necessary to rerun the query. The result of this can then be passed to the API endpoint provided by the partnered supermarkets to build the order. At this point we can update the order table with the date of the *date_placed* and *fee_split*.

Existing and future data
As well detailed above, it will be reasonably straightforward to add this additional functionality to the SSH app. When each user logs onto the app, they will be given a user role with an associated *user_id*. This user role already suffices to guard users access from other users shopping baskets within the household network. Each user will only be permitted to edit the basket belonging to their **user_id**.

The implementation when using the SSH Console Table is slightly more complex, since it may be possible for a housemate to unintentionally add groceries to the wrong users account, causing issues with billing. An SSH camera, pointed at the table can be used so solve this problem.

The washed dishes feature uses facial recognition to determine who is washing dishes at any given moment. This system can be adapted to determine who is using the Console Table when adding groceries, thus adding the groceries to the shopping basket of the person identified by the SSH Camera.

Alternatives

A singular, house-wide shopping basket
My preferred implementation described above allocates each user a personal basket, then aggregating each personal basket to produce the group order. The alternative to this a singular group basket, where each singular quantity of a product is labelled as to who included the item.

- **Pro:** Removes the need for a separate basket relation, since all products are associated with a singular order. This could perhaps simplify the implementation of the feature and jobs associated.
- **Pro:** Removed the need to aggregate individual baskets and produce a combined basket. This will result in faster loading when the group order view is accessed.
- **Con:** Introduces an additional filtering step for accessing individual users baskets. In this implementation, it would be necessary to select only **basket_contains** records where the associated *user_id* matches. Since this will likely be the most commonly accessed feature of the application, this will hinder the speed at which the menu will load.
- **Con:** There may be delays introduced into the search menu as there is an increased dataset to search through to determine if a product is already included in the basket. In comparison to searching only the users basket records, it would have to search the entire house basket.

Updating and caching group basket after every update
This is another possible way for handling the combined basket for the proposed implementation, rather than the alternative implementation just discussed. Each time an edit is made to an individual user basket, this triggers a job on the SSH Hub to aggregate the individual baskets and cache the resulting basket. It is worth considering that the combined basket is expected to be accessed infrequently.

- **Pro:** Almost instantaneous access to the group order when the menu is navigated to. Since the job to aggregate the order is already ran, the cached basket can be immediately retrieved and presented to the user.
- **Pro:** Removes the need for time stamping changes to individual shopping baskets. Since the latest changes are always immediately propagated, there is not need for comparisons to determine whether a cached value is presently accurate.
- **Con:** Each individual basket update triggers a re-aggregation and caching operation for the group basket. In larger households or during busy periods, this could place a heavy load on the system, impacting performance. It is insensible to expend significant resources on maintaining rapid access to a feature infrequently accessed.
- **Con:** The caching logic becomes more complex with frequent updates, requiring mechanisms to invalidate and refresh the cache as needed. This can increase development time and potential maintenance challenges.
- **Con:** If users make multiple small updates in quick succession, it could lead to redundant cache updates and recalculations, which may be inefficient.

Centralised group order manager role
Designate a single user an increased role as 'order manager', given them control of the addition of items for all users, as well as the responsibility for determining the delivery fee split. The concern is that this could increase friction between users if there are disagreements with how the manager handles the cost split and product additions. This could be proposed as an optional feature, however considering the intention of the system to remove points of contention when combining orders, it may not be practical.

Milestones

- **Milestone 1:** Extend the functionality of the current database system to include the three tables *order*, *basket* and *basket_contains*. This gives the necessary data infrastructure to maintain the groceries functionality.
- **Milestone 2:** Develop and integrate a link between the product data supplied by partnering supermarkets and the newly created tables. This should consist of a new products relation including a *product_id* and associated fields. This pipeline should be reusable and update at regular intervals to ensure up-to-date product information.
- **Milestone 3:** Implement a new data type for the shopping baskets.
  - Create new route(s) in controllers for the newly formed groceries component of the SSH system. They should return complete shopping basket datasets with the fields determined in the design overview. The reverse route should also be built to record in the database new products added to a grocery basket.
  - Create storage for the current grocery list within the basket data type to avoid repeatedly running the route to fetch the data.
  - Create a method to aggregate individual baskets and cache that using storage on the SSH Hub.
- **Milestone 4:** Build API queries to post the resulting order to the partnered supermarkets systems. Ensure a get request returns a complete order confirmation.
- **Milestone 5 (Optional):** Integrate facial recognition using SSH Camera to automatically log in a user to their associated account and basket. Use prebuilt facial recognition libraries to reduce workload.
- **Milestone 6 (Optional):** Extend payment gateways to facilitate collecting a divided payment from each user within the order. Ensure checks are in place to confirm total balance has been received before order is placed.
- **Milestone 7 (Optional):** Include functionality to allow individual households to choose appropriate division of delivery fees.

Dependencies

- **UI team:** Will need to design the groceries tab within both the SSH App and SSH Console table for varying aspect ratios and resolutions. This will involve changing existing views to link the groceries page. These designs will then need to be implemented.
- **Database team:** Three new tables are necessary to facilitate the groceries function, allowing for order creation, basket creation and adding products to baskets. They must also develop a product relation in collaboration with the API team to parse product data received from suppliers.
- **API team:** Responsible for handling collaboration with partnering supermarkets infrastructure. They will need to build data pipelines to repeatedly extract relevant product data and make that accessible to the database team. It will be necessary for them to also build API endpoint queries to post orders to the supermarket for fulfillment.
- **Partnered Supermarkets:** We are relying on our partners to provide us with accurate, detailed data to integrate into our system. The level of functionality of our system depends largely on the quality of data supplied.
- **Legal team:** Need to review our legal agreements with customers and supermarkets to ensure that the additional features do not expose us to any new liability. They should focus their efforts on our service of handling payments on behalf of the household.

Cost

There are implementation of this system where there are no operating costs. If we choose to add functionality for each user to pay for their groceries separately, then there will some financial processing fees associated with that. It may be possible to adopt the system for splitting third-party bills used by the Console Table.

If SSH chooses to implement this system, we must be cautious to collect full payment from the house before placing the order. This reduces any risk of non-payment and cashflow concerns. If we accept this money via debit/credit card, this will involve fees. Our options are to absorb these fees as part of the SSH Cloud membership, or add a small service fee to each order placed to cover the card payment fees.

If student households do not wish to pay this service fee, they will be be able to pay for the order independently.

It may be possible however to adopt the system for splitting third-party bills used by the SSH Console Table.

Risks

Since SSH acts as an intermediary between the students and the supermarkets, any issues arising from the supermarket's end, such as delays in deliveries, substitutions, or price changes, may reflect poorly on SSH. This could lead to customer dissatisfaction, negative feedback, and potential liability for resolving these issues. It will be important within our legal terms to clearly declare that we are not liable for any dissatisfaction with the delivery service provided by patterned supermarkets. It would also be advised to give customers access to their order account so that, in the case of order problems, they are able to appropriately escalate this with the supermarket.

Errors in order aggregation, such as incorrect quantities or misplaced items in the group order, could lead to disputes among housemates. This may result in dissatisfaction with the service and potential complaints to SSH. It is imperative that before we enable to service for customers that is it thoroughly tested for this type of error.