SSH – Shared Groceries

Engineering Design Review

Author: Fearne Townsend 2558290

Date: 22/10/2024

Introduction

We have identified that many students order grocery deliveries rather than commuting to shops in person. These deliveries can get expensive if each student within the household places an order separately. Students may opt to save on delivery costs by planning to order their groceries together. However, this requires each student to either curate a list and send it to one housemate to place the order, or to be next to the housemate when the order is placed. This can be quite inconvenient for the students as each one is likely to have a unique timetable of lectures, therefore meeting up is unlikely and often results in multiple deliveries being made regardless.

At SSH we have the opportunity to offer a solution to a problem that students are dealing with within the safety of the app they presently use. Each student within the household will have the app installed, as well as having purchased the console table and SSH Hub. We have the opportunity to extend these to allow students to place group delivery orders, from anywhere, at any time.

We propose to extend the app and build a grocery list page where each student can add items, remove their items or edit their items. This page will track which house-member has added which item, as a result we will be able to track individual bills to facilitate bill splitting between students. The delivery cost provided will be split equally between users who have contributed to the order.

Here we can promote our partnered supermarkets: Aldi, and Sainsburys for users to purchase from, providing information on their stock, prices, and promotions. This gives us the opportunity to build a greater partnership as we funnel our customers towards their stores and increase their sales through our own client base.

Adding this feature aligns with our aim to improve student living and make an already difficult period of their lives easier.

Goals and non-goals

- **Goal:** Create a new page that allows students to create orders, join orders, check-out or delete orders.
- **Goal:** Keep track of which students have entered an order, the items they have added and their individual total cost.
- **Goal:** Update the total order cost with each item, applied promotions and split delivery fees between students. If a promotion applies to the total cost of the basket, split the saving equally between active users.
- **Non-Goal:** Facilitating takeaway orders from local restaurants.
- **Non-Goal:** Placing the order in app. The app will act as a method of adding to the shopping basket remotely, however payment must be made through the console table.

Design Overview

The new Grocery List page will extend the main app and consist of a table of orders, organised from most recently created to oldest. We will keep information on up to 5 created lists to prevent this page from being cluttered and to save memory. The information provided here will be the name of the order i.e. Order 1 as default, or some custom name created by users. It will also display the date the order was placed, changing its status from *Open* to *Closed* . Once an order is closed, it can no longer be edited or amended, only viewed. Orders may also be deleted if the users no longer need them.

To create a new order, one member must navigate to the "+" icon to create a new order. Here they will be able to create a name for the list and select one of our partner stores to order from. We can use their stored address to discover which options are viable for them to receive a delivery from based on locations that deliver near them. We can use the stores metrics for delivery distances.

Each order may be opened and viewed; the information viewable will include:

- Product name
- Individual product cost
- Total number of each item i.e bag of apples x2
- Total product cost
- The username that corresponds to the user that added the item
- Cost per user
- Total order cost
- Applied promotions

For *Open* orders, there will also be the ability to add items, remove items, or edit the list. This will be indicated by a "+" icon to add a new item to the list. This will provide a search of items on the store that has been selected where users can view promoted/discounted items and select the item they wish to add to the list. In the even that the item already exists in the list, increment its value by 1. Each new item will update the individual and total costs based on what each user has ordered rather than an equal value split.

These items will be accessed via data scraping API's that will gather the information from the stores publicly visible sites. There are publicly accessibly API's which we have the opportunity to make use of on APIFY, or we may encourage talks with our partner companies to allow us access to their stock data without having to scrape the information from their sites to improve reliability on stock count and promotions.

In the *Open* orders, members can also search available vouchers/coupons that may be applicable to their order and apply it to their basket.

When the voucher is a total value discount, the savings will be applied to each individuals order i.e for a 20% discount, a user who has £10 worth of items will receive a £2 reduction in cost and a user that has £30 worth of items will receive a £6 reduction in cost. The total bill is £40 with a $8 reduction in total cost, equivalent to the sum of individual savings. This is to ensure a fair split of savings so for potential situations where one user places a much more expensive order than others, they still receive a fair discount and the cost of other users orders isn't being eliminated entirely by the voucher.

This information will be aggregated in table format with the name and image of the item to the left, and product sum, total value, and user who added to the right of each item.

Underneath the table will be a cost breakdown per user, vouchers applied, and the total cost of the order.

To store this information we will create two new tables in our database of each home. *allOrders* and *currentOrder*. These will be connected via key fields where *allOrders* will store the order ID generated by us from a *currentOrder*

allOrders:

Field Name	Data
Order_Number(FK)	Key integer field to link to currentOrder
Order_Name	String Field identifying the order to users
Status	Boolean Value, True(*Open*) or False(*Closed*)

currentOrder:

Field Name	Data
Order_Number (PK)	Key integer field to link to allOrder
Item_Name	String with the name of the item added to the shopping list
Item_cost	Double to 2dp with the price of the item
UserID(FK)	Key integer Field to link to the user that added the item

Existing Data:

Users

Field Name	Data
UserID(PK)	Key integer field to identify user
Name	String containing users name

From the data that we store on Users we will only retrieve their userID, and username as a means to display who has added what item to the order.

New Utility Functions:

As these are new tables that we will need to add, we must also consider new queries to retrieve the data.

Aside from directly returning information from the tables, in *currentOrder* we will also need to calculate the total number of times an item has been added, the total cost of aforementioned item, and produce a list of all the users that have added the item.

Console Table

The additional page will also be accessible through the console table under the icon *Orders*. From here, students can do everything that they can from the app. They can also finalise the payment by

selecting their name in the order to finalise their part. Once all orders have been finalised, payment is made, and the cost is split between users using their already input banking details and will provide delivery updates including the provisional time provided by the supermarket.

Alternatives

Purchase through the app

- *Pro:* This is likely to be more convenient for students as it doesn't require them to be at home to finalise the order.
- *Pro:* It can be linked to a user's Apple pay/ Google pay for secure payment.
- *Pro:* Users without the console table can make use of this feature.
- *Con:* Requires us to consider additional security measures to integrate Apple/Google Pay.
- *Con:* A user can finalise a purchase on their own, even if the entire order is not yet complete.
- *Con:* If implementing the feature where each user must confirm their order is ready before placing the order, each student must confirm the payment simultaneously, using Apple/Google Pay.

Alternative discount allocation

There is an easier implementation of spreading discounts between members, this is by calculating the overall savings on the total cost and splitting the monetary saving equally between contributing users. This however has the potential to backfire as there may be situations where one user contributes significantly more to an order and receives a lower percentage reduction in cost compared to other contributing users, with the potential for the cost of other contributing users orders to be 0 or negative dependent on the savings earned.

Milestones

1. Design the frontend UI to match the rest of our app and integrated with the console table.

2. Implement backend to allow adding to basket, editing orders, confirming order
   a) Integrate our partner stores online order system for students to access real time stock counts, discounts and promotions
   b) Create the tables to store information on previous and current orders as well as queries for accessing the information from these tables
   c) Adjust the "split the bill" feature on the console table for grocery orders to use the suggested method of students paying for their own items + equal share of delivery costs.

3. Integrate our partner stores delivery notification system into the console table notifications to ensure this can be viewed from the console table.

Dependencies

- *UI Team:* Create the new page in app and on the console table to view and edit orders, ensuring that the console table supports all the features within the app.
- *Database Team:* Adding the new tables required to store information on current and previous orders as well as generating queries to access the relevant information from each
- *Notifications Team:* Providing notifications through the console table regarding the delivery
- *Legal Team:* Review our legal agreements with users to ensure the payments made to our partner companies do not expose us to any new liability.

- *Cyber Security Team:* Ensure there are no new vulnerabilities created through the payment feature

Cost

There may be increased costs associated with storing additional data in our databases regarding previous orders. This is why previous orders should be limited to up to 5 most recent previous orders for users to check what they frequently order without unnecessarily storing data on irrelevant orders. We may also offer students to delete previous orders we have stored if they wish.

Additionally, the suggested API's have an associated cost per month of ~£30, this may be avoided if we decide to scrape the data ourselves without the use of another API. This price may also increase if we source information directly from the stores with their permission, however this gives us the opportunity to negotiate as we are driving student traffic towards their stores for purchases.

Privacy and security concerns

All users within the household have already provided their banking details for other orders which we have securely stored. Therefore, there is no need to account for safely handling transactions through the console table, however, we must ensure that the payments made to our partner companies are transferred safely. And continue to monitor the security of payments made through our console table.

Risks

Risk	Mitigation
Users do not wish to purchase a console table and SSH Hub-First class for these orders	Implement payment through the app
Student financial information becomes vulnerable while making the purchase	Closely work with the cyber security team to ensure there are no vulnerabilities for threat actors to make use of or leak student financial information
Cost incurred by storing previous order information becomes too high to maintain	Remove orders once delivery has been confirmed
There are no partner stores near the location of users that support delivery	Expand our partner stores to include a wider range of shops to make this feature accessible to a wider range of users and increase choice for users that have access to all partner stores

Supporting Documents

Ultimate ALDI Scraper · Apify

Sainsbury's Scraper · Apify