Engineering Design Review

SSH - Shared grocery ordering

By Diba Malikzadeh

Introduction

Students living in shared housing often rely on grocery delivery services, especially in recent years as online shopping has become more prevalent. Ordering groceries online offers added convenience, with exclusive discounts and promotions. However, it can also come with many challenges, such as delivery or small order fees, and sometimes minimum spending requirements, making it challenging for students to stay within a budget. Additionally, many discounts or promotions offered online require a high spending tota to apply, which can be difficult for individual students to meet, or can even lead to spending more than needed.

A practical solution to address these challenges is for housemates to place shared orders. This can save money by splitting delivery costs and accessing discounts that require a minimum spending. However, this can be inconvenient, as it requires organising a group. One person would be required to coordinate with everyone else, put together everybody's orders and manage payments. This manual process can be time consuming and cause frustration.

To solve this problem, we propose extending the Student Smart Homes (SSH) app to include a *shared grocery ordering feature*. This will allow students to collectively add items to one shared order, see the total cost, track who ordered what, and calculate each student's individual expense. This solution aligns with SSH's goal to improve the quality of students' daily lives, by making household management tasks easier and more convenient, as well as promoting cost efficiency, which is often a priority for students living away from home.

Goals and non-goals

- **Goal:** Develop a shared grocery ordering fe ature within the SSH app, allowing students to combine orders, split delivery fees, and access bulk discounts. The aim is for 30% of students to use this within the first year of release.
- **Goal:** Allow in-app payment, enabling each student to pay for their own share directly through the SSH app. This should reduce manual money handling, and targets to reduce payment disputes by 50% in the first year.
- **Non-goal:** Promote up-selling or provide product recommendations. Our goal is to simplify the process of making shared orders, not to drive sales.
- **Non-goal:** Develop or integrate new payment systems or processors; this feature will utilise existing payment systems within the SSH app.

Design overview

The shared grocery ordering feature extends SSH app's existing functionality, adding group ordering capabilities via a new button. This button opens a page based on the user's current order status:

1. Creating or Joining a Shared Order
If the student is *not* currently in a shared order, the button will be labelled "Join or Create a shared order", and open a page with options to either:

- **Create a New Shared Order:** The app generates a unique code for the new group order, which the user can copy and share to other students.
- **Join an Existing Order:** Students can enter the unique code provided by another student to join their order. If the code is invalid, an error message prompts the user to try again.

2. Viewing and Editing the Shared Basket
If the student *is* already in a shared order, pressing the button opens the group basket page. This page includes:

- **Shared Order Details:** Students can view the entire order, individual contributions, and the total cost.
- **Item Management:** Students can add or remove items from their individual basket within the group order.
- **Cost Breakdown:** Each students' section displays their items, individual total, and their share of the delivery fee, which is calculated as:

$$Delivery\ Fee\ per\ Student\ =\ Total\ Delivery\ Fee \div Number\ of\ Students\ (n)$$

The basket will be organised into *n* dropdown sections, each corresponding to a particular student. Each section displays:

- The student's name, order items and individual total.
- A *payment status marker* that turns green once the student has checked out and paid.
- The student's share of the delivery fee.

At the bottom of the basket, a checkout button will enable each student to pay for their own order individually. When pressed, students will be redirected to the payment page to complete their payment. The order is finalised once all students in the group have checked out and paid. Upon completion, the app notifies each student that their order has been successfully processed.

Students can *leave* the group order at any time before completing checkout. If they leave, their name and items are removed from the group basket, and their costs are deducted from the overall total. Additionally, the delivery fee is recalculated dynamically for the remaining students. In cases where a student has already checked out and paid, the app will automatically handle any delivery fee adjustments due to changes in the group, notifying students if any additional payment or refund is required.

Backend Structure
The backend will expand SSH's app existing grocery ordering system, introducing *new database tables* to support group orders and collective payments, as outlined below:

Table	Relevant fields	Relevance
`shared_order`	`order_id, created_by, creation_time, status, total_cost, unique_code`	Supports group ordering, by tracking each order's creator, status, total cost, and a unique code that other students use to join.
`shared_order_items`	`item_id, order_id, student_id, quantity, price, added_time`	Tracks items that are added to the shared basket, referencing individual item ID's, quantities, prices and timestamps.
`student_contributions`	`student_id, order_id, individual_total, delivery_fee_share, payment_status`	Managing each student's contribution to the order, including their individual cost, payment status, and delivery fee share, allowing easy recalculations when students leave or join.

The backend automates the *lifecycle* of a shared grocery order, from creation to finalisation, as follows:

1. **Order Creation/Joining:** Upon initiating a new shared order, the backend creates an entry in the `shared_order` table, generating a unique join code for group members. When a student joins, the backend validates the code and updates the `student_contributions` table, recalculating the delivery fee and notifying students of changes.
2. **Updating the Order**: Each time a student adds/removes an item from their order, the backend updates `shared_order_items` and adjusts the total cost, as well as recalculating the student's individual total in `student_contributions`.
3. **Checkout and Payment:** Payments will be processed individually through SSH's existing payment methods. Upon checkout, each student's payment status updates in `student_contribution`, and the order's status changes to `finalised` once all students have paid. The backend processes any refunds or additional payments as needed.

To maintain backend efficiency, daily *scheduled jobs* will automatically close any incomplete or abandoned orders after a defined period, preventing unused orders from accumulating. Additionally, we aim to uphold *data privacy and security* by ensuring only authenticated students can join or modify shared orders, with access restricted to users with the unique group code.

Alternatives

Group admin model
Instead of generating a unique code that students can use to join the shared order, we could assign one student (the creator) to be the *group admin*. The admin would be responsible for inviting other students, approving join requests, removing students from the group, and other administrative controls.

+ **Pro:** Reduces the risk of unauthorised students joining, as the admin can manually approve or decline requests to join.
+ **Pro:** Improves accountability and communication, as one person is responsible for the group and can take charge to ensure the order is processed smoothly and on-time.
- **Con:** Adds responsibility onto one student to assemble and manage a group, which goes against our aim to *automate* the shared ordering process and minimise user effort.
- **Con:** The process of students being added manually, or requesting to join and waiting for approval, adds complexity and could be more time-consuming.
- **Con:** Introduces dependency on a single admin, which can reduce the convenience and automation that this design seeks to achieve.

Handling individual payments
We have considered various approaches for processing each student's payments individually. One main alternative was to process all payments only after all students have confirmed that they are ready, and the group order is finalised. This approach could feel more secure to the students, as payment will be made only once the order is confirmed. However, this method adds an extra step, requiring students to first indicate that they are ready and then return to complete the payment.

Our chosen approach offers flexibility, allowing students to pay for their share whenever they are ready, streamlining the process. In occasional cases where the order is cancelled or fees change, we will ensure that any necessary refunds or additional payments are handled promptly.

Delivery fee calculation
Another way to split the delivery fee among students is to base it on each student's individual contribution to the group order. This way, a student with a higher order total would pay a larger share of the delivery fee than a student with a lower total. However, we determined that it's more advantageous and simpler to split the delivery fee equally among all students, as this promotes fairness and makes all participants feel equal. Additionally, delivery fees are typically fixed regardless of how much you spend, so our approach aligns with standard practices and reduces unnecessary complexity.

Milestones

Milestone 1: Develop the basic *backend structure* needed for shared ordering, including creating the *database tables* (`shared_order`, `shared_order_items`, `student_contributions`). This allows us to test basic functionality for creating, joining

and managing group orders using test data. If performance issues arise, adjustments can be made before proceeding.

Milestone 2: Implement the *frontend components* into the SSH app to support group orders, including the button to create/join shared orders, the group basket page, and payment status markers. Conduct initial usability testing to gather feedback on functionality. Based on feedback, interface adjustments may be made as needed.

Milestone 3: Integrate individual *payment processing and refund handling*. Each student should be able to checkout and pay independently, with the system managing refunds or additional payments if required. If any issues arise, we will refine this logic to ensure a smooth payment and refund process.

Milestone 4: (optional) Implement a real-time *notification system* to alert students of key updates (e.g., when someone joins, leaves or completes payment). Notifications will be tested for accuracy and timing. This milestone will proceed based on available resources and user demand.

Milestone 5: Conduct *extensive testing* to verify all components (backend, frontend, payments, notifications) function seamlessly together. Test data and a small pilot group will be used to identify and resolve any final issues. If no significant issues are found, the feature will be approved for deployment.

Milestone 6: Deploy the new shared grocery ordering feature in the next SSH app release and monitor initial performance and user feedback. Promptly address any issues reported post-launch.

Dependencies

- **UI Team:** Responsible for designing and implementing new UI components for shared ordering, including the "Join/Create a Shared Order" button, group basket view and payment status markers.
- **Backend Team:** Must set up the required database tables and integrate payment and refund handling with SSH's existing systems.
- **Database Team:** Responsible for managing migrations for the new database tables and optimising queries for order management.
- **Payment Team:** Verifies that individual payments and refunds work smoothly within the existing payment system.
- **Notifications Team:** *(if applicable)* Creates templates for any notifications sent to students, such as order confirmations or payment status updates.
- **Legal Team:** Reviews legal agreements to ensure no new liabilities are introduced with shared ordering.
- **Privacy and Security Team:** Ensures compliance with privacy standards and secures shared order data through encryption and access controls.

Cost

We do not anticipate significant increases in operational costs from implementing the shared ordering feature. The backend processes for shared orders, payment handling,

and notifications will leverage existing infrastructure with minimal additional load. Any added data storage requirements are expected to be manageable within current resources.

Privacy and security concerns

The shared ordering feature only uses data that is already accessible to students within the SSH app, such as order details and payment information, and does not introduce any new data collection. Therefore, no new roles or permissions are required, as access is limited to authenticated users who are part of the shared order.

As this feature involves shared financial responsibilities, extra attention is given to securing order and payment information. Payment data will leverage SSH's existing encryption protocols, preventing interception and ensuring secure transactions. Access to group order data is restricted through authentication and authorisation checks, following the principle of least privilege. Additionally, all access requests will be logged for monitoring purposes.

Risks

Risk	**Mitigation(s)**
Students may prefer placing their own order instead of participating in a group order.	Make shared ordering an optional feature. Students can continue to place individual orders as usual through the SSH app.
Students may not use the shared ordering feature.	Promote the feature through in-app notifications, highlighting benefits like reduced delivery costs and group savings. Consider a brief tutorial for first-time users.
Changes in group composition (e.g., students leaving the group) may cause order adjustments and confusion.	Implement clear notifications to inform students of any changes in the group, including adjustments to the delivery fee and refunds, if applicable.
Increased system load from concurrent shared orders and real-time updates.	Optimise backend operations for concurrent group orders by implementing request caching and load balancing to handle high user volumes efficiently.
Unauthorised access to group order information could compromise user privacy.	Limit access to each group order to authenticated users only, using role-based access controls. Regularly review access logs to address any suspicious activity.
Students from different locations may join the same group order.	Enforce a single delivery address for each order, prompting students to confirm this address when joining from various locations.

Supporting Material

- OWASP Foundation, 2023. *Authorization Cheat Sheet*: Best practices for managing permissions, protecting shared data, and ensuring secure access control within applications. https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html