SSH – Grocery Coordinating System App

Engineering Design Review

Author: Mohammed Fizan Shamjith

Date: 30 October 2024
Status: Completed

Introduction

Student smart home is a startup which focuses on creating software and hardware solutions that simply student life in shared accommodation. The many smart home products include the SSH Hub, SSH Cloud, SSH App, SSH Camera, and SSH Console Table.

Students encounter many problems when living in shared student accommodation including dealing with unfamiliar roommates, sharing kitchen space, and managing different schedules. They must also cook and clean for themselves under budget constraints which require them to spend money efficiently, carefully select delivery options and maximize discounts to save for other expenses. Most students usually buy groceries individually, but some flatmates coordinate bulk orders to share delivery fees. This saves time because it avoids trips to the store and money because of split delivery fees and delivery app discounts.

However, coordinating groceries in a flat with many people can be very challenging due to varying schedules and priorities, some want groceries delivered immediately, while others are fine waiting a few days. It's hard to track who adds what to a shared cart, especially when using one person's phone. Splitting costs becomes harder with discounts and remembering individual orders. When deliveries arrive there's a risk of flatmates accidentally taking each other's groceries.

This problem is important to solve because it affects every student living in university accommodation as it helps them save time and money. We propose introducing a new group ordering feature within the SSH App and Console Table to tackle the hassles of coordinating grocery shopping among flatmates. This feature will allow students to:

- **Form Groups**: Create groups with their flatmates using invitation codes.
- **Schedule Orders**: Set deadlines for when orders should be placed.
- **Shared Cart**: Each member adds their groceries to a shared cart, with items and costs displayed under each person's name.
- **Automatic Cost Splitting**: Delivery and service fees are divided equally among group members.
- **Automated Payment Processing**: Individual costs are deducted from each user's in-app wallet at the time of order placement. Users get notified when their wallet balance is low and are reminded if they add items exceeding their balance.

This makes it easier for students because they don't have to rely on one person paying upfront for the groceries and avoids the hassle of repayments allowing students to pay for their own items. This grocery coordinating feature aligns with the main goal of SSH of making life easier and cheaper for students so that they can spend time and money on enjoying their university experience.

Goals and non-goals –

Goal: Implement a shared grocery cart feature in the SSH App and Console table that reduce individual delivery fees by 30% within 6 months of implementation. This can be measured by comparing the average delivery fee paid by the user before and after the feature's launch.
Goal: Automate the payment process so that all group grocery orders are processed without requiring any single person to pay upfront. This is done by integrating a split payment system that deducts each user's share from their in-app wallet when the order is placed. Can be measured by checking the percentage of processed group orders that are successful without payment issues.
Goal: Notify users to top up their wallet when it's below a certain limit or when they add items exceeding their balance, so that orders can be completed without hassle

Non-goal: Handle the actual logistics of the delivery or tracking the delivery
Non-goal: Include supermarkets that don't have an existing partnership with SSH
Non-goal: Recommend personalized grocery based on their previous purchases

Design overview

The grocery coordinating system extends the existing SSH App and Console Table interfaces by introducing a new feature called "Group Order" on the home page of the grocery delivery app. This feature allows users to create or join groups with their flatmates, simplifying the process of ordering groceries together.

SSH App and Console Table (Front end):

- **Group order interface**: Users can create groups and generate unique invitation codes to share with flatmates. Flatmates can use these codes to join the group within the app.
- **Choice of supermarket**: Group members select from partnered grocery stores. To minimize delivery fees, it's recommended to order from a single store per group order unless they want to pay for 2 or more delivery fees
- **Order scheduling:** Users can schedule an order, which will notify other users in the group that an order is scheduled. The users can choose whether to opt in or out of the scheduled order.
- **Shared cart:** The users add items to their individual carts, which is then merged to the shared cart when "Complete Order" is clicked, the entries are recorded in the CartItems table. The system aggregates the user's selected items, records which user added each one, and recalculates the total costs. The shared cart will display each user's name, their items, and individual costs along with the group total. The delivery and service fees are automatically divided equally among the participating users.
- **Payment processing:** At the scheduled time, the system calculates individual totals and deducts funds from users' in-app wallets. Users with insufficient funds receive notifications to top up their wallets.
  The total cost for an individual is calculated by:
  Sum of item prices + (Delivery Fee / Number of Participants) + (Service Fee / Number of Participants)

SSH Cloud (Back end):

- **Group Management Service:** When a group is created, a unique GroupID is generated to identify the group in the system. A random unique code is also generated and stored in the database linked to the GroupID which is used by users to join the group. The membership of the flatmates that join the group is recorded in the Group Members table
- **Order scheduling:** New orders create entries in the Orders table with scheduled times which is recorded by scheduled-time. Users are notified of the order and Participation status is tracked in the OrderParticipants table.
- **Supermarket API:** The SSH Cloud communicates with partnered supermarkets via their APIs to fetch product catalogs, including item names, descriptions, prices, and availability. When an order is placed, the SSH Cloud sends the order details to the supermarket's ordering system through the API. The supermarket then returns the order confirmation, delivery estimate, and tracking information to the users.
- **Notification service:** Sends push notifications for new group orders with scheduling details and prompting users to confirm their participation. Notifies users of low wallet balances or payment issues, prompting them to top up. When an order is confirmed, it notifies all participating users with order and delivery status.

Existing and new database tables that support the group ordering feature:

Table	Relevant Fields	Relevance
Users	user_id, name, email, wallet_balance	Stores user information and wallet balances for payments.
Groups	group_id, group_name, invitation_code, creator_user_id, created_at	Manages group information for group orders, including the invitation code for joining and the user who created the group.
GroupMembers	group_id, user_id, joined_at	Links users to groups
Orders	order_id, group_id, scheduled_time, status	Records scheduled orders and their statuses.
OrderParticipants	order_id, user_id, participation_status	Tracks user participation in orders.
CartItems	cart_item_id, user_id, order_id, product_id, quantity, price	Stores items added to the shared cart by users.

Transactions	transaction_id, user_id, amount, transaction_type, timestamp	Logs all financial transactions
Notifications	notification_id, user_id, message, type, sent_at	Records notifications sent to users.
Products	product_id, name, description, price, availability	Stores product information from the supermarket APIs,

Existing Utility Functions

- **Authentication Service:** Uses SSH's existing user authentication for secure access.
- **In-App Wallet System:** Uses the existing wallet for processing payments and managing user balances.
- **Notification Service:** Extends the current notification system to include events related to group orders.

Roles and Permissions

- **Group Creator:** Has permissions to create groups, schedule orders, and manage group membership.
- **Group Member:** Can join groups using invitation codes, participate in orders, and add items to the shared cart.
- **Access Control:** Implemented through role-based permissions within the SSH Cloud, ensuring users have appropriate access levels.

WebSockets are used to establish a persistent connection between the SSH App/Console table and the SSH Cloud server. This allows for real time synchronization, so all changes made to the shared cart are immediately reflected for all other users. Product data from the supermarket's API is cached to improve performance because it reduces the number of API calls, so users can access the store quickly.

Alternatives

Collecting payments at checkout instead of using in-app wallets:

Instead of making users top up their in-app wallet, we could collect payment directly from users at the time of order placement, using a mobile payment service like apple or google pay.

- Pro: Users are not required to maintain a wallet balance, making transactions simpler
- Pro: Most users are accustomed to paying at checkout like in most other delivery apps and platforms
- Con: Users must be available to confirm and pay at the scheduled time, which may be inconvenient due to varying schedules.
- Con: Users who forget or are unable to pay punctually may delay the entire order processing.
- Con: Students may prefer to control their spending by topping up a fixed amount; paying at checkout could lead to overspending.

Using a Shared Payment Method for the Entire Group

Implement a system where one user, such as the group creator, pays for the entire group order, and other group members repay that user outside the app via cash or bank transfer.

- Pro: Only one payment transaction is needed per group, this makes processing orders faster and less complex.
- Pro: Eliminates the need for In-App wallets.
- Con: Requires a high level of trust among group members, as one person bears the initial cost.
- Con: Managing repayments outside the app can be inconvenient and may lead to delays or conflicts.
- Con: Users may be reluctant to pay the entire cost upfront or rely on others to reimburse them, reducing participation.

Developing a Standalone Group Ordering App Separate from SSH

Create a new app focused on group grocery ordering, separate from the existing SSH App and Console Table interface.

- Pro: Could establish a unique brand identity and attract a broader market beyond SSH users.
- Pro: Could design features and an interface specifically optimized for group ordering
- Con: Will be very resource and time intensive as it requires building an entirely new application.
- Con: May discourage users who are reluctant to download and use a new app.
- Con: Loses the benefits of integrating with existing SSH services, such as shared authentication and wallet systems.

Milestones

Milestone 1: Implement the backend logic for group creation and management, including the necessary database tables (Groups, GroupMembers) and APIs for creating and joining groups. This will allow us to test the functionality with sample data to verify that groups can be created, and users can join successfully.

Milestone 2: Develop the backend functionality for scheduling orders and tracking user participation, including creating the Orders and OrderParticipants tables. Test with sample data to evaluate the performance of scheduling and participation features and identify any implementation issues. Decide whether to proceed based on the results.

Milestone 3: Create the frontend interfaces in the SSH App and Console Table for group creation, joining groups, and scheduling orders using data from the backend developed in Milestones 1 and 2. Deploy a temporary UI to a select group of users to gather feedback. Based on the feedback, either continue development or adjust the design

Milestone 3a: Coordinate with the notifications team to create content for group invitations, order scheduling alerts, and wallet balance notifications. Implement backend changes to send these notifications using our existing notification infrastructure.

Milestone 3b: Allow the design team to create suitable designs for the new group ordering interfaces. We can show the final design to users to gather feedback and make necessary adjustments.

Milestone 4: Connect the SSH App with the supermarkets APIs to fetch product information and handle order placements. Implement the shared cart feature, allowing users to add items, assign them to specific members, and calculate individual costs. Conduct internal tests to ensure the shared cart operates as expected and orders are processed correctly. If any issues arise, we'll refine the integration before proceeding.

Milestone 5: Implement payment processing using the existing in-app wallet system, deducting users' wallet balances when an order is placed and logging transactions appropriately. Test with sample transactions to ensure payments are processed accurately and securely. If there are significant, we may need to reconsider this approach.

Milestone 6: Implement real-time synchronization of the shared cart using WebSockets, allowing users to see updates immediately when others add or remove items. Test the performance and scalability of this feature. If it becomes too resource-intensive or does not enhance the user experience, consider removing it

Milestone 7: Work with the design and development teams to finalize the user interface and experience. Ensure all new features are integrate seamlessly with the existing SSH App and Console Table interfaces. Conduct thorough testing. If user feedback is positive, we can deploy the new features in the next SSH release.

Dependencies

Frontend/UI Team: Will design the new group ordering interfaces in the SSH App and Console Table. This includes creating screens for group creation, joining groups, order scheduling, and shared cart management to ensure real-time updates and a consistent user experience.

Backend Development Team: Will update the database schema to include new tables necessary for group ordering, implement backend logic and APIs for group management, order scheduling, shared cart functionality, and integrate with supermarket APIs. They will also handle payment processing through the in-app wallet system

Notifications Team: Need to develop and implement notifications related to the group ordering feature, such as group invitations, order scheduling alerts, participation confirmations, low wallet balances, payment issues, and order status updates. They will integrate these notifications into the existing infrastructure.

Security and Compliance Team: Will review the new group ordering features to ensure it complies with data protection laws and payment regulations. They will assess the system for potential security vulnerabilities, ensure sensitive data is encrypted and that standards like PCI DSS are followed.

- **Legal Team**: Will review and update legal agreements with supermarket partners to cover the new feature. They will ensure that all new features comply with relevant laws and regulations, protecting the company from potential liabilities.

Cost

We do not anticipate any significant increase in operating costs as a result of implementing the group ordering feature. The additional server load from real-time synchronization using WebSockets is expected to be minimal due to group sizes usually being small. Data caching will reduce the frequency of API calls to supermarket partners, keeping external costs low. Overall, our existing server infrastructure and cloud resources should be sufficient for the new feature without requiring many changes or increased expenses. If we encounter performance issues, we could consider optimizing synchronization methods.

Privacy and security concerns

The group ordering feature uses existing user data and only shares order details to group members who have joined the same group. No additional personal information is collected, and current roles and permissions effectively manage access control. All new APIs will adhere to the best security practices, including input validation and protection against threats like SQL injection and cross-site scripting (XSS). WebSocket connections for real-time updates will be secured to prevent unauthorized access. Payment processing will meet PCI DSS standards, keeping financial data encrypted and safe. Appropriate audit logging will be implemented for key actions within the group ordering system, using our existing audit event framework to monitor and detect any unauthorized access.

Risk

Risk	**Mitigation(s)**
Students may not use the group ordering feature	Promote feature within the app, provide tutorials, encourage usage through discounts and gather user feedback for improvements.
Real-time synchronization increases server load and causes performance issues.	Improve code and make it more efficient, monitor server performance and scale server resources if necessary
Payments fail due to lack of funds in users' wallets, causing orders to fail.	Implement pre-order balance checks and notify users to top up wallets
Security vulnerabilities can be introduced by new features which expose user data.	Ensure our systems are secure by following security practices, regularly check for issues, and keep all data encrypted and protected
Relying on supermarket APIs can lead to service disruptions if APIs are unavailable.	Implement caching strategies and maintain communication with supermarket partners to manage changes.
Group members can have disputes over cost sharing or order errors.	Make cost calculations transparent, clearly display individual items and allow users to review their order before confirmation.
Legal issues from sharing user data among group members without proper consent.	Ensure users provide consent when joining groups and update terms of service and privacy policies to comply with data protection laws.

Supporting material

- Kevin Sookocheff, 2019. How Do Websockets Work? https://sookocheff.com/post/networking/how-do-websockets-work/
- OWASP Foundation, 2023. *Cross Site Scripting (XSS)* - https://owasp.org/www-community/attacks/xss
- OWASP Foundation, 2023**.** *SQL Injection* - https://owasp.org/www-community/attacks/SQL_Injection
- PCI Security Standards Council, 2022. Payment Card Industry Data Security Standard: Requirements and Testing Procedures. Version 4.0, March 2022.