OpenKart 2.0 : Collaborative Shopping

Akshay Jain
NC State University
Master of Computer Science
Raleigh, North Carolina
Email: anjain2@ncsu.edu

Jithin John
NC State University
Master of Computer Science
Raleigh, North Carolina
Email: jjohn3@ncsu.edu

Surabhi Sudame
NC State University
Master of Computer Science
Raleigh, North Carolina
Email: ssudame@ncsu.edu

Zaheen Khan
NC State University
Master of Computer Science
Raleigh, North Carolina
Email: mkhan7@ncsu.edu

***Abstract*—With the advent of technology, online shopping is considered more of an essential than a luxury. However a major hurdle that most consumers face today is that shipping or delivery charges for a small one time order is prohibitively high and coordinating and planning grocery shopping to meet the minimum order limits of most online shopping platforms is a hurdle. To solve this problem people tend to collaborate with their friends or buy unnecessary things just to exceed the free shipping amount. At times, people altogether cancel the plans to buy some nonessential goods or make the trip to the brick and mortar store to avoid these extra charges. OpenKart v1.0 aimed at providing a platform for the consumers of online shopping applications to collaborate while shopping that could be helpful in gaining monetary advantage by employing crowdsourced logistics. OpenKart v1.0 is an innovative tool that helps with the cumbersome task of finding people in nearby area and collaborating to place a common order delivered to a commonly agreed upon location and timeline thus either entirely bypassing or sharing the delivery charges. OpenKart v2.0 builds upon this solid platform to enhance the user experience to make the tool more appealing to the user base, by adding a series of features aimed to provide a great user experience.**

***Keywords*—** Collaborative shopping, Android app, Notifications, Firebase,GeoFire, GeoLocation, NoSQL, SMS, authentication, Realtime chat

I. Introduction

Online shopping has become an integral part of the lives in the twenty first century. Online content hosting platforms and traditional shipping companies have formed a network that moves billions of items across the face of the planet on a daily basis. The shipping companies uses advanced algorithms that can best optimize delivery routes and timings to economically move packages. However this becomes difficult in a several cases like grocery shopping and rural areas with lower frequency of deliveries. The shipping optimization systems face the biggest challenges in predicting shipping patterns or coordinating shopping schedules of people. This results in higher cost in shipping part of which is levied as a fees on the consumer. One idea this solution attempts to take advantage of, in order to solve this problem is the power of crowd sourcing. The idea is to provide a platform to empower the consumers to anticipate, communicate and coordinate the shopping activities. This allows a crowd controlled platform that can coordinate the shopping activities to best utilize the shipping facilities to best place the order. This allows the consumers to make monetary gains by coordinating the shopping activities to incur least shipping charges. These solutions also helps in solving problems in commuting to grocery stores without cars or while buying things online and not be able to reach upto free shipping amount.

This takes a lot of trouble and sometimes it happens that people dont check the group or arent members of group and thus bear some loss by ordering something extra or paying the shipping amount. This solution is not well coordinated and also it takes a lot of time to gather total number of people necessary to reach the free shipping amount.

II. Literature Review

It is a well-known fact that people do not like to pay extra charges beyond what their items cost. Online shopping is a convenient way of getting our work done without the hassles of commuting and paying for the commute. If the shipping charges are beyond expectations, the overall sales enhancement due to internet may be limited. Studies have shown[1] that shipping and handling fees are the number one factor driving shopping cart abandonment.

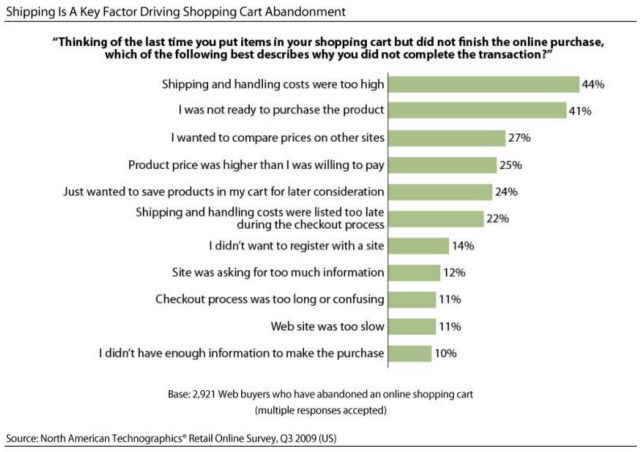

Fig. 1: Study on different issues[1]

As we can see in figure 1, shipping and handling costs were a major factor in people abandoning their cart. This causes inconvenience to the user and loss of market to the shopping entity.

For people online looking to collaborate while shopping online have very limited choices today here is a study of how

various existing methods compare against each other is factors that makes them suitable for the problem.

Research to find similar platforms yielded no results of close matches, except Amazon Manage your Household feature[5] which allows friends and family to share a common amazon subscription. We have attempted to present the results of the comparative study that involved this Amazon feature along with few other methods we think can be alternatively used to collaborate while shopping online. The conclusions of this study included parameters that we believed was most relevant for a suitable solution for the problem of collaborative shopping with logistics. These are presented in the fig 2. Here are conclusions from this study.

Platform/ Method	Possible Collaborators	Ease of Bookkeeping	Privacy & Security	Multi Platform Support	Ease of Communication	Shopping Experience
Social Media	+++/	/- - -	/- - -	+++/	+++/	/- -
Amazon Household	/- - -	+/	++/	/- - -	/-	+++/
Share accounts on Walmart.com etc.	/- -	/- - -	/- - -	/-	/-	+++/
Offline collaboration with friends & family	/- -	/- -	+++/	++/	/- - -	/- - -
OpenKart 2.0	++/	+++/	/-	++/	++/	+/

Fig. 2: Study on different issues[1]

1) Social Media - While using social media platforms to collaborate while shopping online scores highly on the ability to reach large masses of potential collaborators and also gives you freedom to shop on any platform, while at the same time giving you possibilities of communicating on several platforms. But this method fares really poor on factors like data management where there is no dedicated ways to manage data on the orders, like whos ordered what, the amounts owed by the collaborators and status of the order. Privacy and security are also low on these platforms since you have to share disproportionate amount of private data to stay connected and collaborate on a shopping order. And the shopping experience is also scored low, since you there is no easy way to find details on shopping platforms and then share it with collaborators.
2) Amazon Household - This feature by amazon allows adding additional users to a single subscription in a secure manner, and this allows shopping on the original platform and at the same time collaborating on a common cart. But this has serious disadvantage of only being able to reach limited users as there is no dedicated features to find potential collaborators, nor does it provide any robust bookkeeping features when there are multiple collaborators. Also this does not provide any communication platforms between collaborators. It does score very high on shopping experience and provide high security, but privacy is not scored high since we need to share the entire subscription providing access to extra details like email id/username etc. Also to communicate with collaborators it would require sharing further private details like phone number etc. And currently this feature is only available on Amazon among the most commonly used platforms limiting flexibility to shop on multiple platforms.
3) Sharing Online Platform Accounts - This is not a feature that adds anything other than shopping experience to the social media method while scoring low in almost every other factor thus making it a not very desirable feature.
4) Offline collaboration with friends and family while reducing privacy and security concerns reduces scores in almost every other option over the social media method.
5) OpenKart v2.0 - This provides a dedicated platform for shopping collaboration thus providing access to a large number of potential collaborators, though not in

the same number as social media platforms, this is still a large enough number of serious collaborators, and also has the advantage of better finding suitable collaborators, using search open carts feature. OpenKart v2.0 being custom made application for shopping collaboration provides a set of features to manage data on the orders, amounts, items, collaborators list etc, and also features like smart orders and merge carts allows to collaborating really easy. OpenKart v2.0 uses firebase authentication feature to provide a secure platform. It also tries to maintain privacy by not requiring to share private contact details to communicate by providing built in chat feature thus allowing easy and secure communication. But due to ability of users ability to still see all orders and items and delivery locations of all orders, OpenKart v2.0 score poorly on data privacy. OpenKart v2.0 also provides support for any online platforms although it provides enhanced features like smart orders only for limited online applications. So even though it scores higher on multi platform support than other options it still loses some score over this limited support to smart orders. Shopping experience even though is lower than shopping original platforms such as Amazon and Walmart, it has scored quite higher than social media platforms, the new smart order feature has significantly improved shopping experience since now collaborators can shop without leaving OpenKart 2.0 but at the same time be able to see most details including pictures of products from the original platform.

III. User Study

We had sent out a survey form to assess the extensions that we want to make to the already built OpenKart 2.0 system. Some of the questions and the analysis we have come up with is as follows:

A. Which notification or alert method would you prefer for this app?

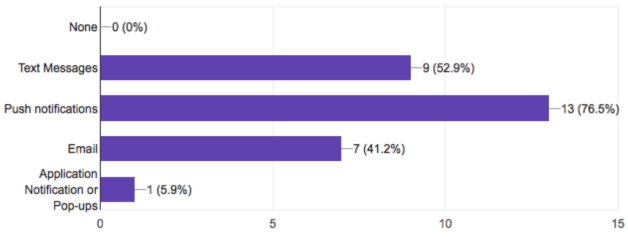

Fig. 3: User Review on Alert Methods

We found that the current application's scope did not include sending any kind of notifications to the user, which we thought that could add an edge to the user experience and make the app much easier to use. Thus we asked users this question to find out which kind of notification method would be most preferred by the users. The results as represented in figure 3, are really straight forward that the best way to notify users would be push notifications. This helps us to apply our resources in the one of these notification methods as each of these require a significant time in integrating and testing each of these notification modes, which prevents us from including all the modes of notifications in this single effort. We have decided to concentrate our efforts in designing and enhancing the application to include push notifications.

B. Would you prefer to get notifications or alerts when there is a change in the contents of the cart?

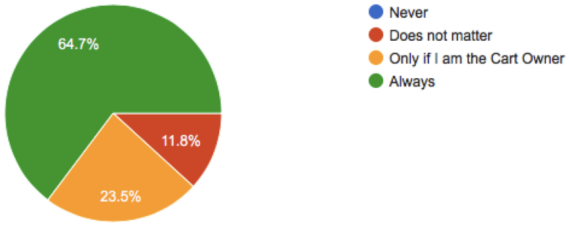

Fig. 4: Notification Preference of Users

Since we have recognized that adding notifications to OpenKart 2.0 is important, we asked this question to users to figure out when the users would want to receive a notification as we all know that extra or unnecessary notifications is something that absolutely no one wants. Figure 4 helps conclude that About 25% of the users would want notifications only when they are the cart owners and about 65% would like to receive them all the time irrespective of them being the cart owners. Even we feel that the users need to be updated all the time since their orders are also involved. This helps in tailoring the default settings of the notification feature, which help avoid negative user experience leading form unnecessary pestering notification.

C. How comfortable are you about sharing your online order details with other users?

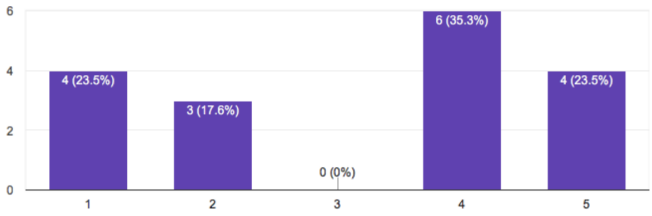

Fig. 5: User Reviews on Privacy of Cart

This question was motivated from a privacy concern point of view. We have asked this question to decide whether the items of a users cart should be made private or not. As Figure 5 shows, We have received really mixed opinions from users about how they feel about sharing their order details with fellow users of this application (who may even be complete

strangers). As we have received really mixed opinions, we have decided to not move forward with this as one of our feature enhancements.

D. Would you prefer having a payment method integrated with the application?

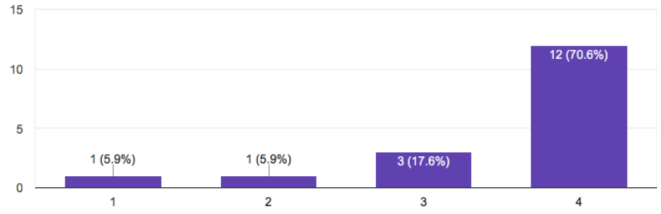

Fig. 6: User Reviews on Adding a Payment Method with App

We asked users if they would like to have a payment method integrated with this cart sharing application, and more than 90% of the users feel that is a great idea and would really love to such a feature with OpenKart 2.0 . Figure 6 shows that a lot of Users would like to have this app integrated with a payment method. Although even we feel that this is an interesting and helpful feature for this app, we have decided to not go ahead with this feature extension as we think that to implement this feature, our security has to be increased multifold as it involves a lot of sensitive data like the users bank or credit/debit card details. We feel that this does not fit the current scenario and timeframe.

E. Would you like to have the app place your order directly?

Fig. 7: User Reviews on Automatic ordering

As figure 7 shows, around 40% of users are not comfortable or indifferent towards the the app auto-ordering on their behalf and also as it is not possible given our current aim and resources. We have decided to make the ordering process simple by helping and easing their process of cart creation rather than placing orders on the users behalf.

F. What are the important feature according to you that should be added to this application ?

We have given users a set of features and asked them to rank them into top 3 according to what they think. The list of features given to the users were:

- Notify cart owner when target amount is reached.
- Integration with a payment method to share cost among collaborators.
- Suggest the second cart creator to automatically combine the contents into another cart that is already available. (Automatic Cart Merging)
- Get alerts when a new cart in available in the specified radius.
- Integrate with expense tracking apps like Splitwise.
- Integrate with online shopping websites for automatic orders.
- Allowing orders to be marked Urgent, and optimize suggestions based on Urgency and preferred collaborators

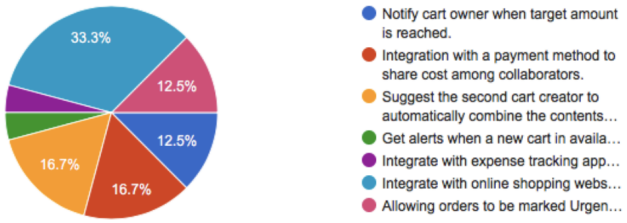

Fig. 8: Most Important Feature According to Users

As figure 8 highlights, the first choice of most of the users was to integrate it with online shopping websites so that the process of transferring or replicating the cart from the app to the website will be tedious process. The user will have to keep switching applications and then searching and selecting the items in the cart from the app on the online shopping website. This makes the user indulge in useless extra efforts.

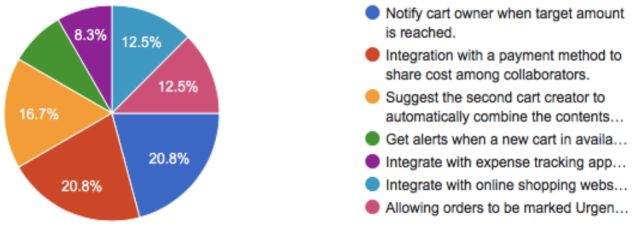

Fig. 9: Second Most Important Feature According to Users

As figure 9 shows, the second most important features according to our surveyors are providing the users with notifications and integrating a payment feature into the application. Even according to us, notifications to users should be enabled as this will increase the user experience greatly and as this app includes users buying things and eventually paying for it, we feel that the user should be continuously updated.

Although we acknowledge and understand that payment integration is an important feature with regard to the purpose to this application, we have already discussed why integrating a payment feature is not feasible for us right now given the scope and time frame of this project.

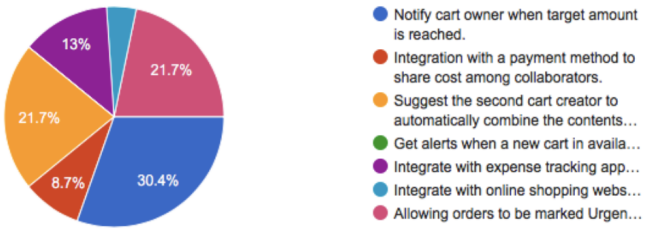

Fig. 10: Third Most Important Feature According to Users

As highlighted by figure 10, Users see that making orders of each user private and the automatic cart merging as the other important features that we should be adding to OpenKart 2.0 .

We have decided to exclude making orders completely private as this will create complications when replicating the cart from the app to the website as if the order items are kept private then how will the cart owner know what items to add into the actual cart on the shopping website. Thus we have decided to implement a part of this as we know that users value this feature. The orders will be made private to everyone else other than the cart owner as this is the only possible solution to this issue as of now.

We also want to implement the auto-merging of carts as a feature as we think that users fail to recognize that deleting their own cart and re-adding the items into someone elses cart is a very tedious and time consuming method and will eventually discourage users from merging their carts and will slow down the target completion and users will simply have to compete between one another.

IV. Existing Solution

The first version of the application has been developed by group B as part of the Phase 1 project of Software Engineering course. This mobile application is well developed with modular code base along with the extreme necessary features. A user can create a cart setting up the target, add his or her own items, and wait for other users to add their items and reach the specified target. Once the cart reaches its specific limit, the users can chat amongst themselves to decide on the time and place of delivery and collaborate on picking up their items from the place of delivery. Before creating a new cart, a user can check for existing carts nearby using a convenient slider for radius adjustments. If found a cart that the user is comfortable with, the user can add his or her items to the cart along with their price, the link of the food store for the item, etc. The price of his or her items is then added to the cart and once the total price reaches the target amount, the users can collaborate on the chat window and place the order to get their items respectively.

V. New Features and Enhancements

Our time constraints could not permit us to implement all the features discussed. After an open discussion amongst the team, we agreed on implementing 2 major features and 2 minor features to enhance the application by an improved scale:

A. Auto-fill Product details

As mentioned above, since popular stores do not have open APIs that expose their data, our best bet right now is to scrape data from their websites. We will save this data on our database servers and auto-fill the product details for the users.

- Feature description As the user starts typing a product name, the user is shown a drop-down list consisting of all the products from the store which the cart belongs to, starting with his or her typed letters. The user can click on the product he or she wishes to add to the cart, and the URL and price are automatically filled for the user. The default count autofilled is 1. Once the user clicks on Submit, the item is added to the cart and the user is taken to the cart details page. The user can verify the item added is the one he or she intended by clicking on the item, as it takes to the original store page for that item. In Figure 11, a product is being added to a smart order for Patel Brothers, and the details of the product were auto-filled by the application on clicking on the product from the drop-down.

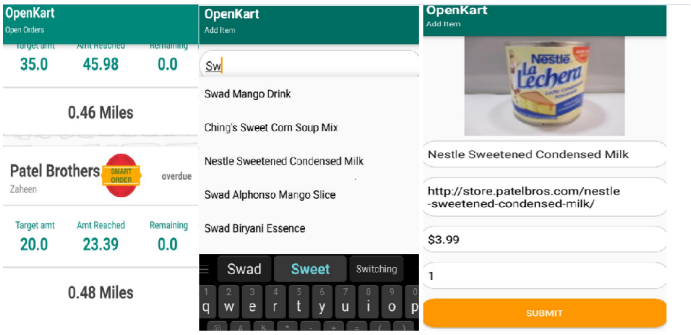

Fig. 11: Smart Orders

- Design Decisions
  - User needs to enter URL or Name There were 2 options for us to auto-fill other details for the user: The user enters the URL and all the other details are automatically filled, whereas the user types the name and sees a drop-down and gets to select the item, whose other details are auto-filled.
    We decided to go for the latter because it is more convenient to the user. If the user has to enter the URL, there is still a need for the user to move away from OpenKart 2.0 application, go to the store website, search for the item, copy the URL, navigate back to our application, and paste the URL. With the name option, all these steps are avoided, and replaced by the user typing in some letters of the item name. However, there is a trade-off here. With this option, the user is expected to know a few letters of at least 1 word in the name of the item, whereas in

the first option, the user can probably search for the expected item in the different categories available on the store website. Though, the user can always fall back to going to the store website and searching for the product the way he or she likes, and copy the name instead of the URL. We also know that most users will at least have some basic idea of the name of most products that they want to purchase.

- Pre-fetch contents v/s Fetch on-demand We were faced with making a choice between pre-fetching product details and storing it on our server versus scraping the website and obtaining this information as the user types the name. We decided to go with the former, because the latter would have needed scraping the website for each letter (or some specific number of letters) that the users type in. This would have degraded the responsiveness of our application. Also, for stores with a large number of products, the time taken to obtain the information would have been proportionate. The trade-off was that if the details are pre-fetched, it would incur storage cost on our servers. With the cost of firebase realtime storage cost of dollar 5/GB, this was a considerate choice.
- Number of default auto-filled product count 1 was chosen as the default count so that if the user overlooks this information and the product is bought by the cart owner, the losses incurred to the cart owner or the adder of the product would be minimal. 0 was not chosen as the default count because then the user would compulsorily have to change the count, which increases a step in adding a product.

- Technologies
  - AutoCompleteTextView An editable text view that shows completion suggestions automatically while the user is typing. The list of suggestions is displayed in a drop down menu from which the user can choose an item to replace the content of the edit box with. The drop down can be dismissed at any time by pressing the back key or, if no item is selected in the drop down, by pressing the enter/dpad center key. The list of suggestions is obtained from a data adapter and appears only after a given number of characters defined by the threshold[6]. For this application, the data adapter was populated by data from the Firebase realtime database, where data for product details was fetched from the store website

B. WebScraping from a third-party store website

- Feature Implementation Since popular stores do not have open APIs that expose their data, our best bet right now is to scrape data from their websites. We will save this data on our database servers and auto-fill the product details for the users. We have implemented a webscrapping prototype for one online grocery ordering sites, PatelBrothers. This PoC can be extend in future for other stores as well. Our goal was to go through each grocery category available on the website as highlighted in figure 12 (screen shot of Patel Brothers website) and store following information of all items available in each category.
  - Product Name
  - Product Price
  - Product Category
  - Product Image
  - Product Link
  - Category Link

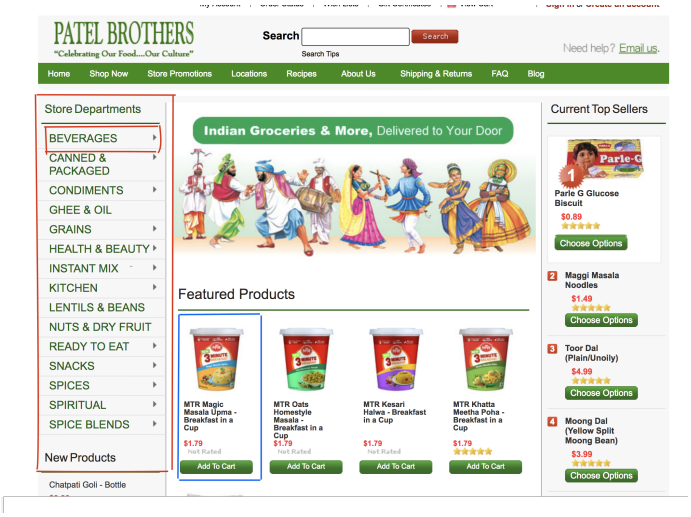

Fig. 12: Third Party Website

There are many tools which can be used to extract data from html pages. We are using well known BeautifulSoup python library for parsing the content of PatelBrother website. BeatutifulSoup pulls data out of HTML and XML files and convert it into a format that is usable for analysis. It is done by creating a parse tree of the page as shown in figure 13. BeautifulSoup provides many ways to navigate, search, and modify this parse tree. Figure 14 shows a part of the tree generated by beautiful soup for PatelBrother Website on local IDE.

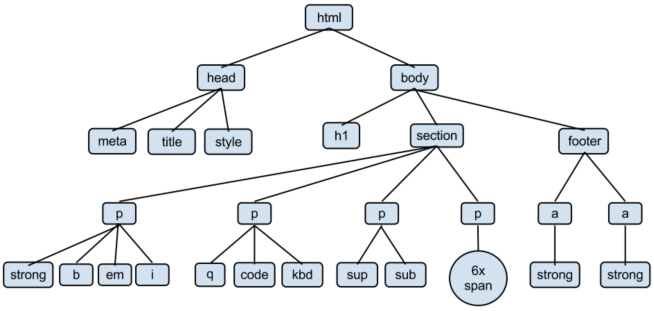

Fig. 13: HTML Parse Tree [10]

Once the Parse tree was generated, we navigated through the tree to analyze the pages. We created a list of tree children and looked for the classes we were interested

in scrapping. First step was to fetch links for all the Categories. To do this we searched for side category bar in our tree, iterated through each category, and stored the category details in a dataframe. Beautiful Soup parses the given HTML document into a tree of Python objects. There are four main Python objects that you need to know about: Tag, NavigableString, BeautifulSoup, and Comment. Searching relevant nodes of this tree is done using select operation provided by beautiful soup. Select search for tags that match two or more CSS classes. The result is then filtered based on id, href and tags. The divisions with categories and category links are filtered by WebScrapper. This result is further parsed and data is stored in the Data Frame.

Fig. 14: Parse Tree for Patel Brother Website

The WebScrapper crawls through each of the category links to fetch content of each category. Each Category page has list of items arranged as blocks. Also, each category has variable number of pages of Products (shown in figure 15). To do this WebScrapper fetches number of pages and links to each page associated with the category as shown in figure 15.

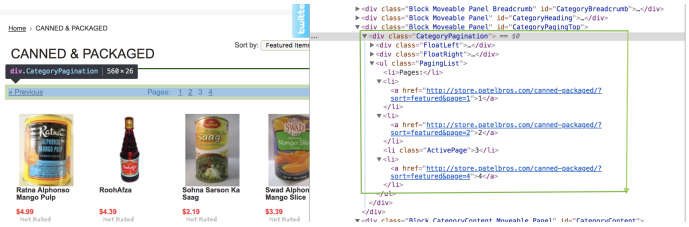

Fig. 15: Page Links Container

WebScrapper recursively goes through each page link and scrape data from each page by creating new html parse trees. We explored the structure of the content page of each category to find the HTML and Class Attribute it uses. To do this we used devtools from google chrome as shown in figure 16 and also analyzed the parse tree output generated by beautiful soup. Analyzing the parsed tree of these pages, we understand that Items are encapsulated under div CategoryClass as shown in figure 16. Soup tree is now searched for this id using select.

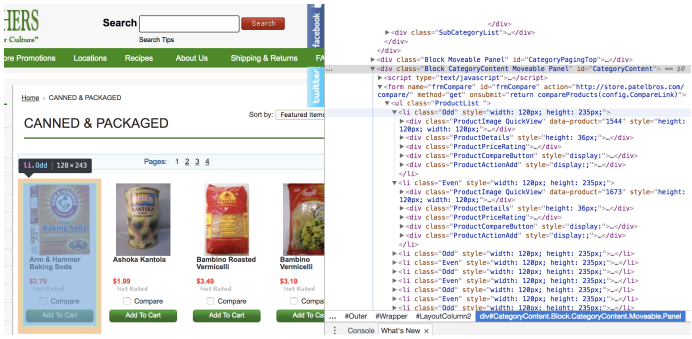

Fig. 16: Product List Container

WebScrapper fetched the html page, and ran html parser on the content page to generate the child tree. Each Tag may contain strings and other tags. These elements are the tags children. Beautiful Soup provides a lot of different attributes for navigating and iterating over a tags children. We are using the find feature to do lookup in the child nodes of the page tree. Beautiful Soup defines classes for anything else that might show up in an XML document: CData, ProcessingInstruction, Declaration, and Doctype. Just like Comment, these classes are subclasses of NavigableString that add something extra to the string. While parsing these elements of the list of children, the script skips the child object of type navigable string, and only look for another BeautifulSoup object. As a parameter, we provide id of the children. We scrape the unordered list of all item for each page. An example of this item node is shown in figure 17.

Fig. 17: Product Container List Element

Finally, each node is parsed, based on the required class to fetch item details, and stored in another dataframe. This dataframe can be written as a CSV or Json, using Pandas APIs. The WebScrapper script to extract content is invoked by another master script. The master script establishes a secure connection to the firebase database using firebase Python client. The key file to connect to the firebase can be downloaded from the firebase account. Then, the master script updates the catalog table in single bulk update passing the scraped data in JSON format. The master script is configured as a cron job on an Ubuntu cloud machine on Google Cloud Platform which is currently being executed every 12 hours.

- Design Decisions
  - Python Packages We evaluated two python packages Scrapy[2] and BeautifulSoup for our WebScrapper. Major difference between the two is that BeautifulSoup is only used to parse HTML and extract data and Scrapy is used to download HTML, process data and save it. As we only wanted to scrape data and store it our data we decided to go with BeautifulSoup. Limiting the functionality and offloading the processing to Firebase made Webscapping very fast with BeautifulSoup. Another reason to use BeautifulSoup over Scrappy is the ease to learn and implement as it is beginner-friendly.
  - Cron job Server We evaluated two ways of setting up cron jobs. Initially, we explored google cloud function, which allowed us to use built-in cron job mechanism within firebase application using JavaScript cron job which would have invoked the Webscrapper. But we had to abandon this approach because we have extensively used many python packages which have c language extension and as of today Google Cloud Function platform, does not support installing Pyhon packages which include a C language extension. So, we had to move to another independent virtual machine where we had the flexibility to install any kind of python packages

C. Merge Cart Option

- Feature description The owner of the cart sees an additional Merge Cart button on the cart details activity. Once the user clicks on the button, other carts are shown that belong to the same store as the original cart. The user has the option to adjust the radius parameter to see carts closer to him or her. Once the user clicks on the cart, the items from the original cart are added to that cart, and the original cart is deleted. The original cart is opened and the user can verify that appropriate items were added to the cart. Figure 18 shows the merge cart feature in OpenKart 2.0.

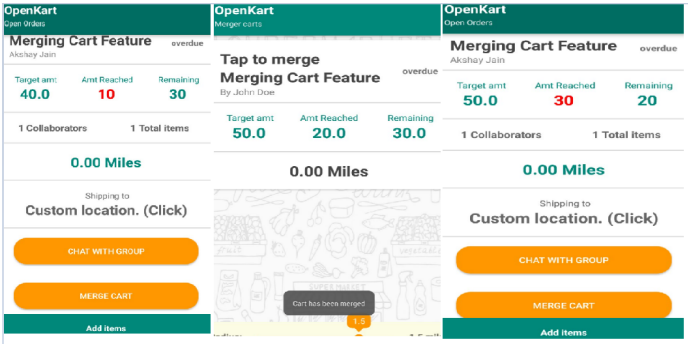

Fig. 18: Merge Cart

  - Cart Collaborators Approval The other users apart from the cart owner added products to the cart based on the original cart details. They may or may not be satisfied by the cart details to which the original cart is being merged to. The cart owner has the right to merge the original cart to the other cart. Should he or she take the other collaborators approval before merging the cart? If yes, how should this approval be taken? We decided that the cart owner can use the chat option for the approval of other collaborators. A better design would be to send a notification to each collaborator and ask for approval, then add the products of only those collaborators who give their approval. However, this was complex to implement. Faith was placed in the cart owners decision and it was concluded that most collaborators would give the approval anyway.
  - Benefit of Merge Cart given the fact that auto-fill is implemented With auto-fill, users can add products to a cart very quickly. With auto-fill already implemented, was it really beneficial to provide the option to merge cart directly? It was noted that time taken with merge cart was 10 folds lesser than time taken to add products again. Furthermore, only the cart owner is involved in merging the cart and the collaborators have no hassle of adding their products again. Thus, the merge cart was seen as an essential enhancement, and its utility was not affected by the auto-fill option.
  - Cart merge without owners/collaborators knowledge We were challenged with merging carts using algorithms based on distance and target amount, and whether it would make sense to do so, rather than giving this right to the cart owner to do himself or herself. It was concluded that proving the correctness of such an algorithm would be difficult unless we have history data. Also, the algorithm could only use distance and target amount as parameters, and not the relationships of people with each other, since this is not captured by our application. Cart owners may merge their carts with someone they know for ease of picking up their products. Also, the user experience wouldnt be great if the cart was merged implicitly without the knowledge of the owner.
- Technologies The merge cart feature was implemented in the app itself, by fetching cart contents and collaborators from Firebase realtime database and inserting them to the other cart. No new technologies were used.

D. Notifications on target amount reached

- Feature description Cart owner gets a notification when the total cart amount reaches or exceeds the target amount. Notification can be seen in Figure 19 where the cart owner was informed that the cart is ready to be ordered.
- Design decisions
  - How many times should the cart owner be notified? We were faced with the following questions: Should the cart owner be notified only once when the

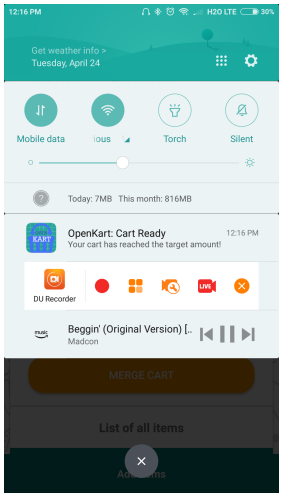

Fig. 19: Notifications

target amount is reached? What if this notification is missed by the owner? Should we sent the notification again? If yes, when and how frequently should this notification be sent? Should a notification be sent every time when a product is added beyond the target amount?
We decided to go with only 1 notification. It was concluded that the best option would be to send a periodic notification to all carts that have reached the target amount. However, due to time constraints, this was not implemented, as it would require the implementation of a cron job separately.

- Notification to cart collaborators on target amount reached It was discussed whether a notification should be sent to cart collaborators when the cart target amount is reached. This was opted out as the collaborators did not have any action to take on receiving the notification. This notification would have irritated the collaborators and might have been a bad user experience. This decision came out of a think-aloud conversation between the team members itself.

- Technologies There are a number of solutions available for sending notifications to Android phones. We decided to use Firebase Cloud Messaging (FCM)[3] realtime database, hosting, authentication, etc.
FCM is a cross-platform messaging solution that lets you reliably deliver messages at no cost.
Using FCM, we can notify a client application that new email or other data is available to sync. We can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4KB to a client app. It's key capabilities are:
  - Send notification messages or data messages Send notification messages that are displayed to your user. Or send data messages and determine completely what happens in your application code.
  - Versatile message targeting Distribute messages to your client app in any of 3 waysto single devices, to groups of devices, or to devices subscribed to topics.
  - Send messages from client apps Send acknowledgments, chats, and other messages from devices back to your server over FCMs reliable and battery-efficient connection channel.

E. Bug fixes

The cart owner name was defaulted to John Doe in the first version. This was changed to the real owners name. Though this bug fix appears to be small, it captures the essence of user relationships. Users might be comfortable with collaborating with other users they already know, and the utility of the application may increase drastic

F. UI Enhancements

- Color codes for carts The amount reached displayed on the cart was color coded. Green was chosen for carts that have already reached the target amount, whereas red for carts that have not reached the target amount. These colors were used because green and red are known to be complementary colors[7]. Also, since users should be encouraged to add to carts that have not reached their target amount yet, red was chosen based on the research that it is the most attention catching color[8].
- Total amount for each collaborator Collaborators can see the total amount of the products added by them to the cart. This allows the collaborator to keep a tab on their expense in the cart without needing him or her to manually add up the costs. This achieves better user convenience.

VI. Software development strategies

Following are some of the software development strategies used by us during the course of working on the second phase of OpenKart.

A. Agile Methodology

Learning from previous experiences, we decided to use the agile methodology to develop the OpenKart 2.0 . We implemented short sprints of a week. At the end of every week we picked up new features for development depending on our progress, importance of the feature and the time remaining.

B. Continuous integration

Although manually initiated, we tried to follow the practice of continuous integration. Every feature was continuously integrated into the mobile application as soon as it was developed, and incremental enhancements were then made.

C. Fail fast

We discussed the idea of the application with a few users early in the development phase to gain feedback, decide features and develop the application accordingly.

VII. Evaluation Results

We created a user survey, and gathered a few people to evaluate the working of OpenKart 2.0 . We asked various questions to the users and asked them to record the time taken for various functionalities of the application.

A. How much time did it take for you to add an item into the cart without using the smart order(Autocomplete) feature?

As mentioned above, the application first used a manual method for adding items into the shopping cart. After incorporating the autocomplete feature, the time taken to add an item significantly decreased. Users were asked to add an item into the cart using the old manual method of adding items and the average time taken to add an item into the cart was recorded. The following chart shows the trends in the time required by various people to add an item the cart.

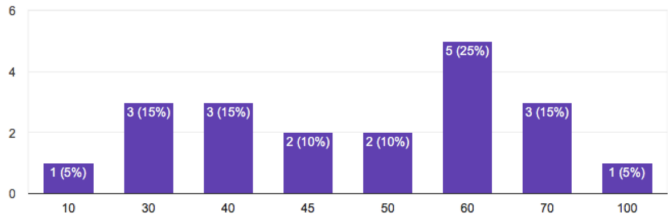

Fig. 20: Time taken to add item without using smart order

Fig. 20 shows that only at a single instance when the user could memorize the name and cost and url properly from the online store, they were able to add an item in 10 seconds. On the other hand almost half of the people took a minute or more to add a single item into the cart. Thus we can assume that the average time taken by users to add a single item was about 45 to 50 seconds.

B. How much time did it take for you to add an item into the cart using the smart order(Autocomplete) feature?

The same set of users were then asked to add items into the shopping cart using the smart order feature, which actually autofills all the details on behalf of the user. Each user was asked to enter an item into the cart without leaving the OpenKart application and the time taken by each user was the recorded and a chart was created out of the data collected.

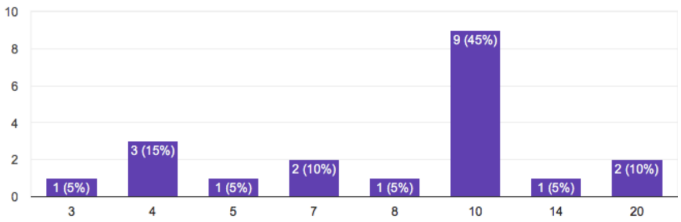

Fig. 21: Time taken to add item using smart order

Fig. 21 shows that the average time taken by users to add an item into their cart took an average of around 10 seconds. Some users were also able to find and add their items in even lesser than 4 seconds. The maximum time taken by any user was 20 seconds which still makes it a very big improvement than the previous method of adding items. The time taken is reduced by a whopping 80%.

C. How much time did it take for you to merge your cart with someone else's cart manually? (in seconds)

We asked the evaluators to merge their cart with another existing cart manually, that means they had to add all the items from their cart into another cart manually. The above results show that manually adding single items into a cart can be so tedious, the results for this question also match those results. The following chart shows the distribution of time taken by users to manually merge carts.

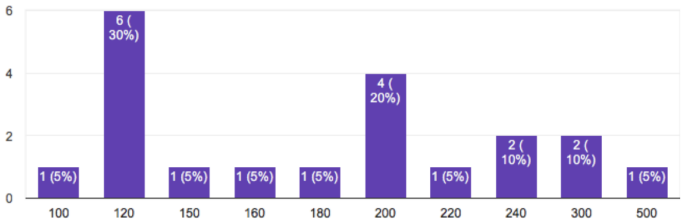

Fig. 22: Time taken to Merge Cart Manually

From Fig. 22, we can see that on an average users took 3 minutes and plus to transfer around 5 items from one cart to another. Users also had to shift pages which made it a really tedious and annoying process. In the next round, we also asked users to also use the autocomplete feature to shift items from one cart to another. This also on an average took users around 30 to 40 seconds to transfer 5 items from one cart to the other.

D. How much time did it take for you to merge your cart with someone else's cart using auto-merge? (in seconds)

We asked the users to then add or migrate their entire cart into another cart. This essentially lets you merge your cart into another cart automatically. The time taken to do this was also recorded and is shown in the following chart.

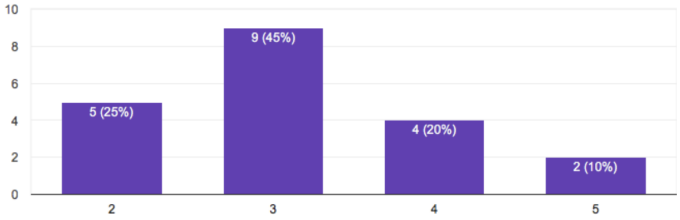

Fig. 23: Time taken to Merge Cart using Auto Merge

Fig.23 shows that merging carts takes no more than 4 to 5 seconds to merge any number of items from one cart into another. This is a very very significant improvement from the previous method that would have been used. As compared to the manual method of merging carts, the time taken reduces by 99% , which is a really good improvement. When compared

to the manual method but using the autocomplete feature, the time taken still reduces by 94% which still is a very good improvement. Thus, although someone may say that the features do overlap a little, they still cause great improvement in performance.

E. How often did you you not find a product through auto-complete that you know is being sold on the website ?

We asked the users if they ever found any items missing in the auto-complete list which they surely knew was being sold on the online store for that particular store. The results collected are plotted on the chart displayed below.

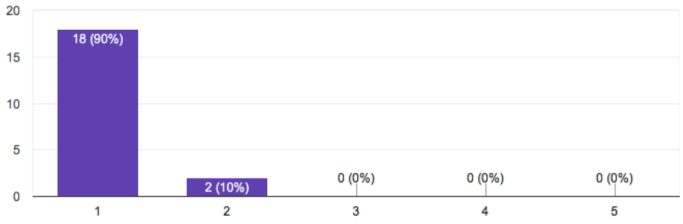

Fig. 24: How often user did not find items

Fig. 24 shows that the users always found the products that they were searching for on our platform. 90% of the users found all the products that they were searching. 10% of the users did not find the product which could have happened because of various reasons. It could be because the server fell down for a while and the cron job did not execute or the item had been deleted from the website or if a new product was added and our scraping algorithm did not catch it as it fell in the 12 hour blind spot. Our application does not store any details of the items that were deleted from the online store thus no message can be even showed to the user. This is a small gap in the current version of the application.

F. Did you verify the auto completed product details for your personal satisfaction by visiting the actual product link?

This question was asked to check if users actually went and verified the links provided by autocomplete to verify the correctness of those links.

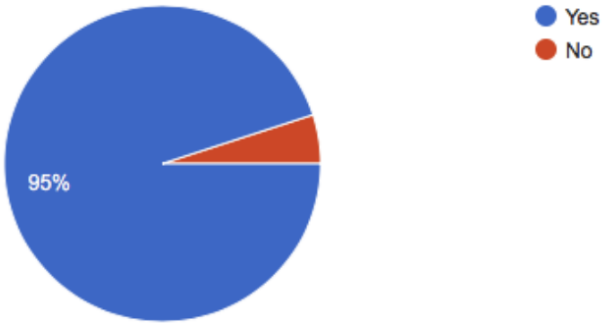

Fig. 25: How many people verified the link

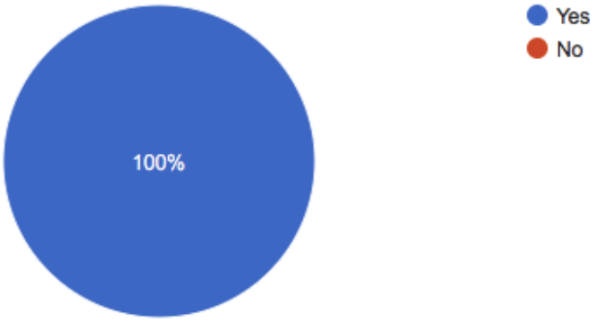

Fig. 26: Verifying Correct Product Details

Fig. 25 and Fig. 26 show that 95% of the users did actually verify if the links provided were valid or not. 100% of the users found valid and correctly working links. Although in the start users may check links to verify that the right products were added, but after a few tries, we expect all users to develop trust on the application as the possibility of it going wrong is really low.

VIII. Future Work

The code is clean, modular, follows DRY principles, and needs simple deployment strategies. Future work can be focused on enhancing the existing features and adding more features, such as follows:

- Shared products Collaborator can be a group of users instead of a single user. This facilitates easy tracking and sharing of expenses amongst the group of users.
- Private carts Some users may only be comfortable in sharing the cart with collaborators they know. Users should be open to mark a cart as private and add collaborators themselves that are allowed to add products to the cart. Invite link can also be integrated and used for addition of collaborators, such as the one used for Whatsapp groups[9].
- Merge sub-cart Owners and collaborators should be allowed to migrate only their products to another cart.
- Group feature Users should be able to create a permanent group of users they are comfortable in sharing their cart with.
- Payment method integration Cart should be integrable with a payment application such as Venmo with account information and have the option to auto-pay when a cart is ordered.
- Urgent orders Users should be able to mark a cart as urgent. This will encourage collaborators to add items to a cart when they are in need to get their products sooner.

IX. Conclusion

OpenKart 2.0 is oriented towards enhancing user convenience and experience. The features like inventory extraction is geared towards making it easier for users to add new items to existing carts and also for the cart owners to easily

place orders using links to the right products. The merge cart feature provides great efficiency in adding a bunch of products together from an existing cart to another cart. Such user convenience encourages more collaboration and in turn increases the utility of the application. Use experience was also another area that we focused on. Notifications were added to engage users with the application while providing useful information that avoids manual checking for users. Some bugs were fixed and user interface was enhanced with color codes and individual total amounts to aid in obtaining context information clearly and easily. Overall the newer version of this app is designed with user convenience and experience at the highest priority.

REFERENCES

[1] *Smarter Strategies For Free Shipping - Understanding The True Costs And Benefits to Retailers*, https://www.ups.com/media/en/Smarter_Strategies_for_Free_Shipping.pdf

[2] *Scrapy - An open source and collaborative framework for extracting the data you need from websites*, https://scrapy.org/

[3] *Firebase Cloud Messaging*, https://firebase.google.com/docs/cloud-messaging/

[4] *OpenKart: Collaborative Shopping*, https://github.com/srujanb/OpenKart/blob/master/MarchFinalReport-OpenKart.pdf

[5] *Amazon: Manage Your Household*, https://www.amazon.com/myh/households

[6] *AutoCompleteTextView — Android Developers*, https://developer.android.com/reference/android/widget/AutoCompleteTextView

[7] *Mobile App Design: 14 Trendy Color Schemes — Adoriasoft_Com — Medium*, https://medium.com/@Adoriasoft_Com/mobile-app-design-14-trendy-color-schemes-2669b5bb77d3

[8] *Seeing Red - Why Red Is a Good Contrasting Color for an Effective User Experience — ICS - Integrated Computer Solutions*, https://www.ics.com/blog/seeing-red-why-red-good-contrasting-color-effective-user-experience

[9] *WhatsApp Gets Shareable Links Feature for Group Chat Invites — Technology News*, https://gadgets.ndtv.com/apps/news/whatsapp-gets-shareable-links-feature-for-group-chat-invites-1467488

[10] *Web Scraping with Beautiful Soup*, http://web.stanford.edu/~zlotnick/TextAsData/Web_Scraping_with_Beautiful_Soup.html

X. REVIEW CHITS

- XJD
- NLU
- IXJ
- MAN
- ALA
- ZAO
- MWL
- DVP
- JXT
- UYB
- MGZ
- AQT
- ZIP
- HUF
- CSF
- WZQ
- HZL
- NOB
- IXH
- ZPY