**Grocery Sharing Application - GROC SHARE**

**Final Project Report - Mobile Cloud Computing.**

**Sriram Venkatakrishnarao | Karthikeyan Swaminathan | Sagar Thakkar**

**(110452166) (110562357) (110533717)**

**Problem statement:**

As students and otherwise, grocery shopping is a pain. For students especially during core deadlines and exams time it becomes imperative to stock the groceries for large period. How if we device a system which facilitates students and others to combine their purchases and students take turns to get stuff.This still is pain on part of some.How if we think of making a system which inputs your requirements and places order by pooling it with others and later your order gets delivered to your door step. Our application does exactly the same. For this application we have considered SBU students as target audience and developed it to make grocery shopping a better experience.

**Abstract:**

Our App named GrocShare is an android based application which is used to merge the orders on the fly and place the orders to the vendors. Generally for getting free order delivery we have to always order something which satisfy the threshold amount.This amount varies from Store(Vendor) to Store. For this application we have taken this threshold as 25$. Now what if i have an order which is less than 25$. I will have to forcibly order something extra just to get my order placed.There are many individuals(students) facing the same problem. What this app does is it facilitates the user to add their orders not worrying about the minimum requirement. The application stores all the orders of different users in cloud and when the total of all such individual orders which are less than threshold combine together to form a cumulative total whose value is equal to or greater than 25$, the app then merges all such orders and places the order to the vendor.

**Key Words:**

Merge Orders, Google App Engine, Google Cloud Messaging(GCM) , UserID,Orders.

**Introduction:**

This section will give brief idea about the various technologies which were used to build the application.

***Android Studio:***

Android Studio is the official IDE for Android app development, based on [IntelliJ IDEA](https://www.jetbrains.com/idea/). On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance productivity when building Android apps.

For developing the Application we are using Android Studio 2.0. While building the apps we have explored and extensively used the features like

1. Using flexible Gradle-based build system
2. Testing the system across various build variants and generated multiple APK files.
3. Used Code templates to help build common app features
4. Used rich layout editor with support for drag and drop theme editing
5. Accessed Lint tools to catch performance, usability, version compatibility, and other problems
6. Extensively utilised Built-in support for [Google Cloud Platform](http://developers.google.com/cloud/devtools/android_studio_templates/), making it easy to integrate Google Cloud Messaging and App Engine

***Google App Engine:***

Google App Engine is a platform as a service (PaaS) cloud computing platform for developing and hosting web applications in Google-managed data centers. Applications are sandboxed and run across multiple servers. App Engine offers automatic scaling for web applications—as the number of requests increases for an application, App Engine automatically allocates more resources for the web application to handle the additional demand.

***Google Sign-In API :***

Google Sign-In is a secure authentication system that reduces the burden of login for users, by enabling them to sign in with their Google account—the same account they already use with Gmail, Play, Google+, and other Google services.Google Sign-In is also a gateway to connecting with Google’s users and services in a secure manner. You can give users the opportunity to pay with Android Pay, share with their Google-wide contacts, save a file to Drive, add an event to Calendar, and more. Integrate Google’s user-centric APIs and services inside your app to help your users take action and convert.

***Google Cloud Messaging (GCM) :***

Google Cloud Messaging (GCM) is a free service that enables developers to send messages between servers and client apps. This includes downstream messages from servers to client apps, and upstream messages from client apps to servers. For example, a lightweight downstream message could inform a client app that there is new data to be fetched from the server, as in the case of a "new email" notification. For use cases such as instant messaging, a GCM message can transfer up to 4kb of payload to the client app. The GCM service handles all aspects of queueing of messages and delivery to and from the target client app.Here's how these components of GCM architecture interact:  
Google GCM Connection Servers accept downstream messages from your app server and send them to a client app. The XMPP connection server can also accept messages sent upstream from the client app and forward them to your app server. For more information, see About GCM Connection Server.  
On your App Server, the HTTP and/or XMPP protocol is implemented to communicate with the GCM connection server(s). App servers send downstream messages to a GCM connection server; the connection server enqueues and stores the message, and then sends it to the client app. If you implement XMPP, your app server can receive messages sent from the client app. The Client App is a GCM-enabled client app. To receive and send GCM messages, this app must register with GCM and get a unique identifier called a registration token. For more information on how to implement the client app, see the documentation for your platform.

**Problem Description:**

Local Grocery Stores have a minimum limits to place orders for free home delivery. Particularly for foreign students to get their regional items it is at most important to group together and order else they end up spending hefty amounts to get the stuffs.This leads to dependencies within students to get their orders through.Also the students and customers may have no opportunity to save from bulk purchases unless they merge their orders.This bottleneck of forming groups forms the real concern.Some current apps have the facility to form the orders and place orders.However the real bottleneck is not merging people but merging orders.If we can make something that has minimum pain to users by placing the order and be relaxed to get the orders at the door steps.Here the technology developed takes input both the minimum threshold and the bottleneck of merging orders and provides a solution by placing the merged order on the fly from backend when it finds other pending order to be served and combined total crosses the minimum threshold value.The next section wiill give details of the solution developed.

**Solution Methodology:**

The android application GroShare is is developed to make grocery shopping from local stores an enjoyable experience. The app incorporates following functionalities :-

1. ***GOOGLE BASED SIGN IN:*** Anybody who has google account can use this application. Google Sign In is used to validate the users and store the details in the backend Cloud server which keeps it to track the orders and keep information to map the orders to the particular User.
2. ***ORDER PLACEMENT:*** The GUI of the map is simple enough to select the items available from the catalogue and place the quantity pertaining the item. Once the item is selected , the user clicks the Add Item button to put the item to the order. The particular of the item details is shown in the text view at the bottom of the GUI. Once all the items are added the user selects Submit button to place the order. On Submitting the order there are always chance that the user might wish to remove a particular item from the order.So on the Confirm Order Screen the user can remove the item by selecting particular item in the LIST view shown. If the Order is final , the user then selects the confirm order button to push the order to the cloud(Vendor).
3. ***CLOUD DATABASE:*** The Order pushed by the user is pushed on to the cloud. The application uses Google App Engine as the Cloud Server and places all the order in the cloud server database.The Cloud receives USERID, ORDER ITEMS,TOTAL which it stores in a database.The database also has a field of Status which the cloud will update on placing the order to the vendor. By default all the orders will have a Pending Status mapped to their orders.It is only after merging that the status updates from Pendng to Competed.
4. ***ORDER MERGING:***  The script in the backend is so built that all the incoming orders are placed in the database with the status of Pending.The Script then queries all the pending orders in the database. The script then combines the total of each of the pending orders. If the cumulative total of the merged order crosses the minimum threshold value of 25$. It then merges the orders and sends the merged order to the vendor with the information of USER ID ,order items , quantity and total of individual orders. If the pending orders total does not cross the minimum threshold it does nothing and waits for another item which can be merged and would cross the minimum threshold.
5. ***HISTORY ORDERS:*** The Backend database server has all the orders. The user can request the history orders from the cloud by selecting the History Order Items in the Navigation Drawer which the app has.On Clicking this the App will send the USER ID to CLoud and request all the orders it has placed. The backend on receipt of this query shall query database to obtain all the order details for that USERID and will send the same to the application.The application receives the response and places it to a table and makes user visible all the history orders.
6. ***UTILITY APPS INTEGRATION:*** The Groc Share app has also given a functionality to open the utility apps like SQUARE CASH and SPLITWISE to settle up the money transactions and updating the order details in the SPLITWISE Group.The navigation drawer has the options of the utility apps which on selection shall start the corresponding applications.
7. ***ANDROID AND GMAIL NOTIFICATION:*** Whenever the order is merged by the backendserver. It places the order to the vendor via an email.At the same time it also sends the information about the placed order via and email and application notifation. We implemented this by using Google Cloud Messaging service and Email IP provide by the app engine.

**Flow Chart :**

***App Side***

GrocShare.png

**Fig 1 : App End Flow Chart**

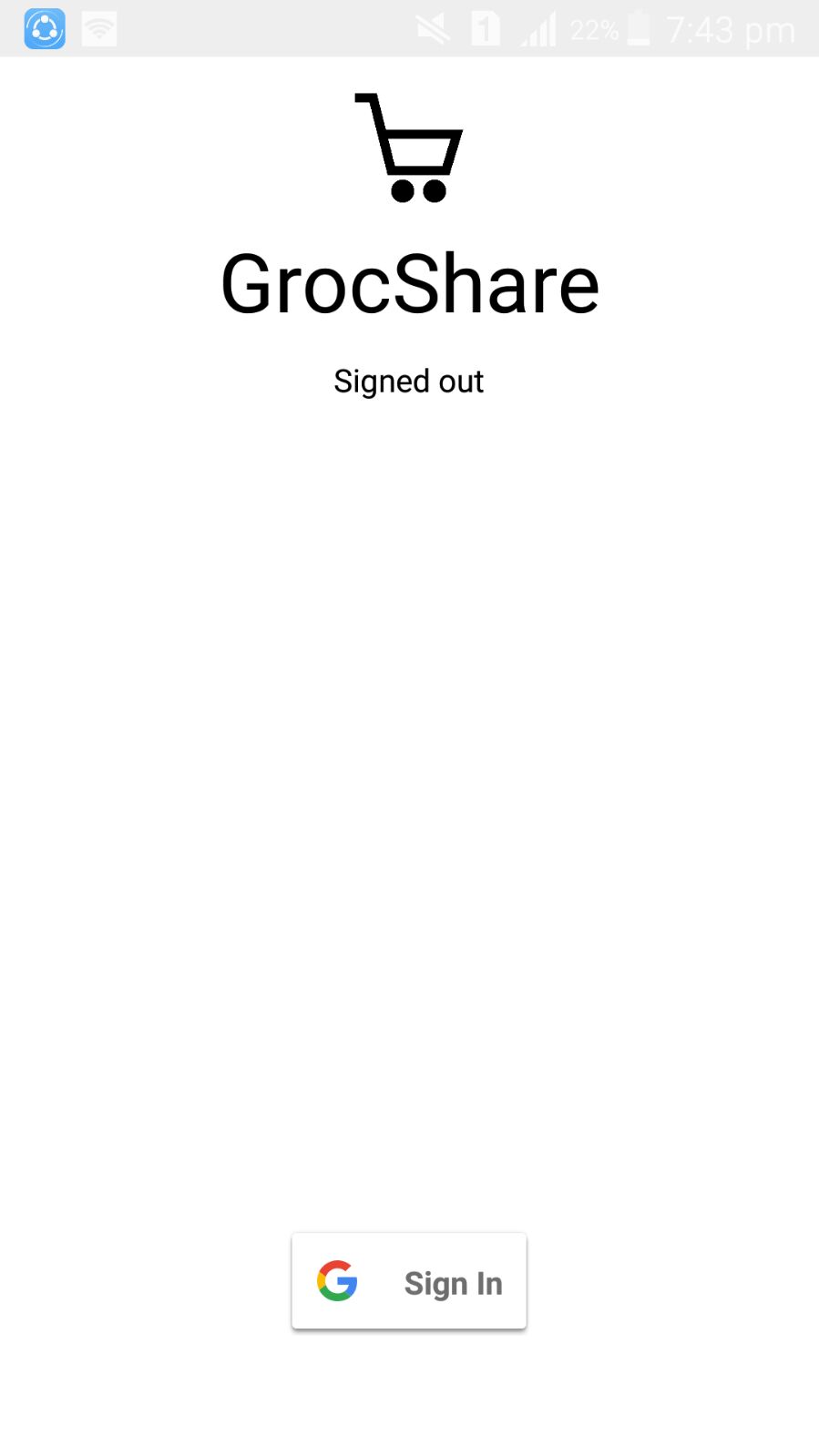
***Cloud Side:***

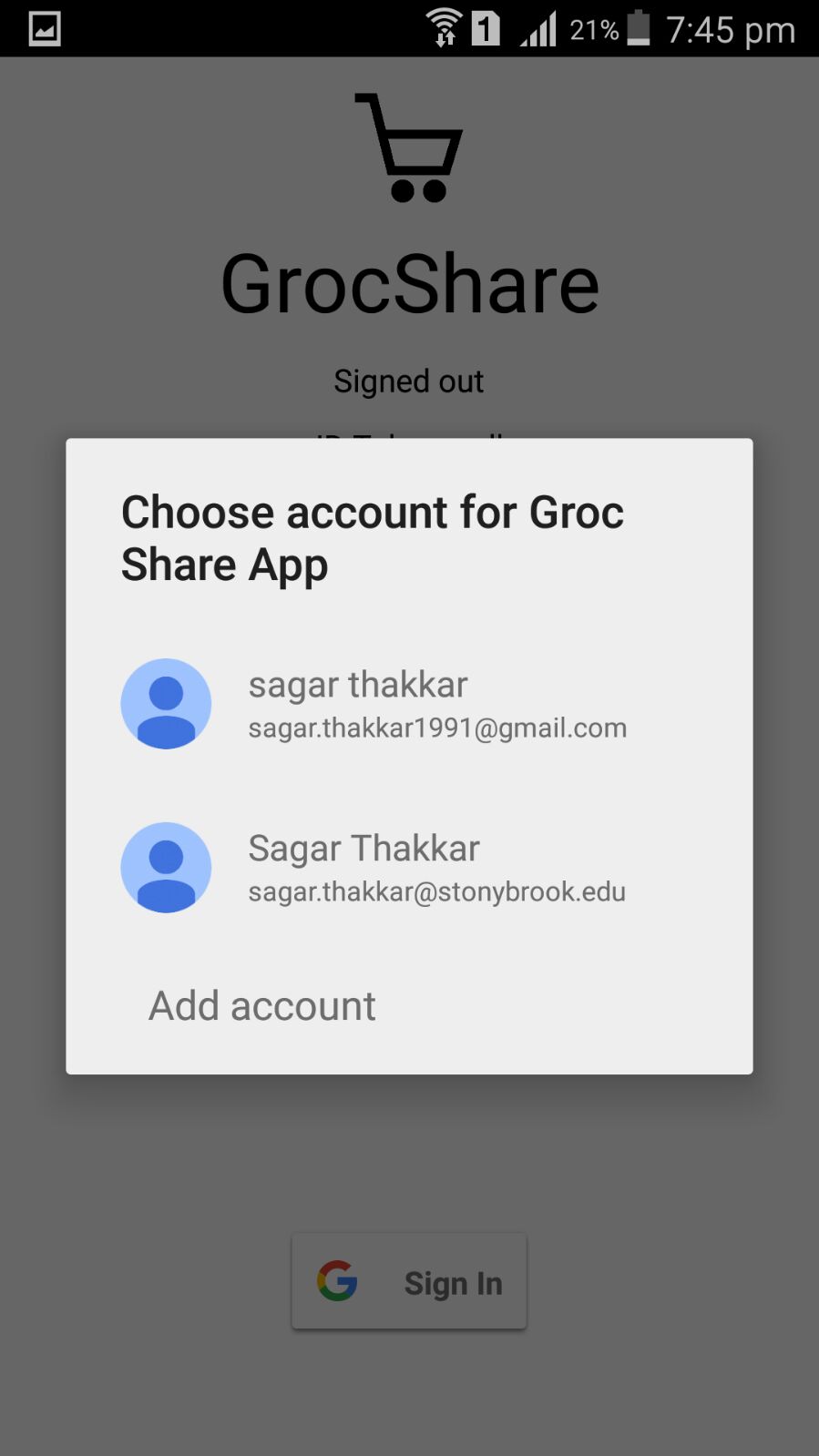
GrocShare Cloud.png

**Fig 2 : Cloud End Flow Chart**

**Evaluation:**

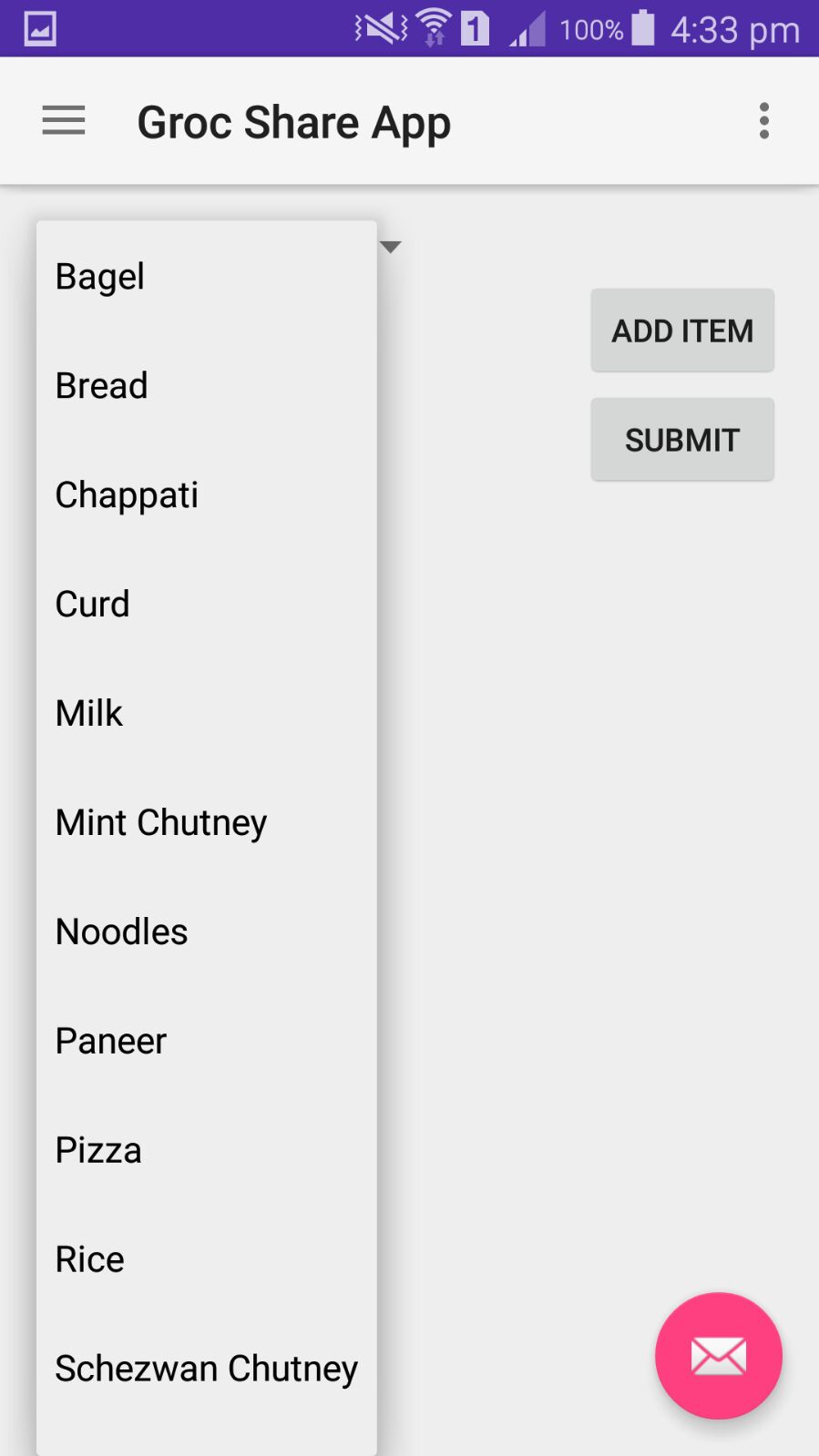
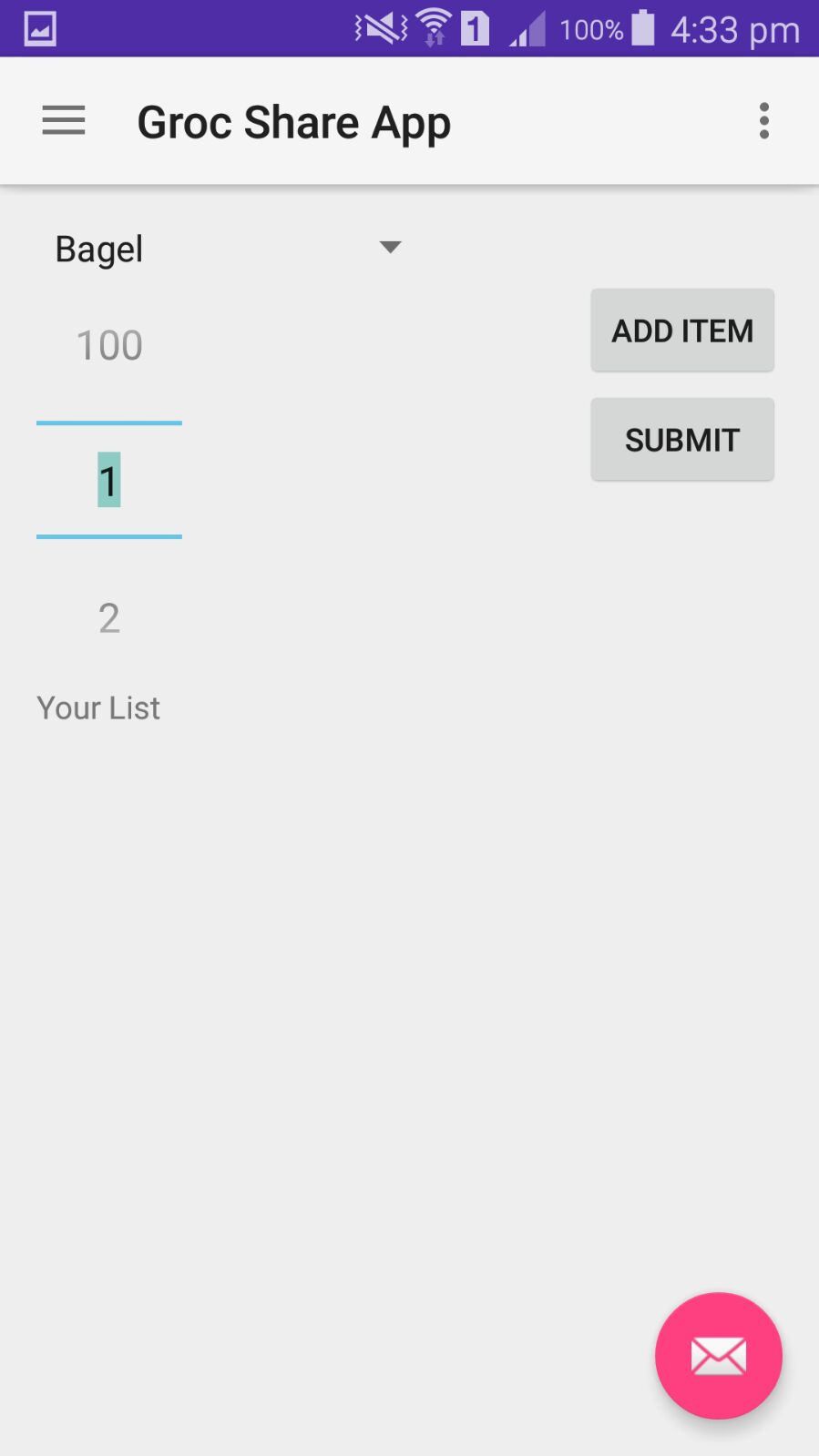
This section gives glimpses of the app and its functionality via a sequence of snapshots which we captured from the app.

***STEP 1: GOOGLE BASED SIGN IN***

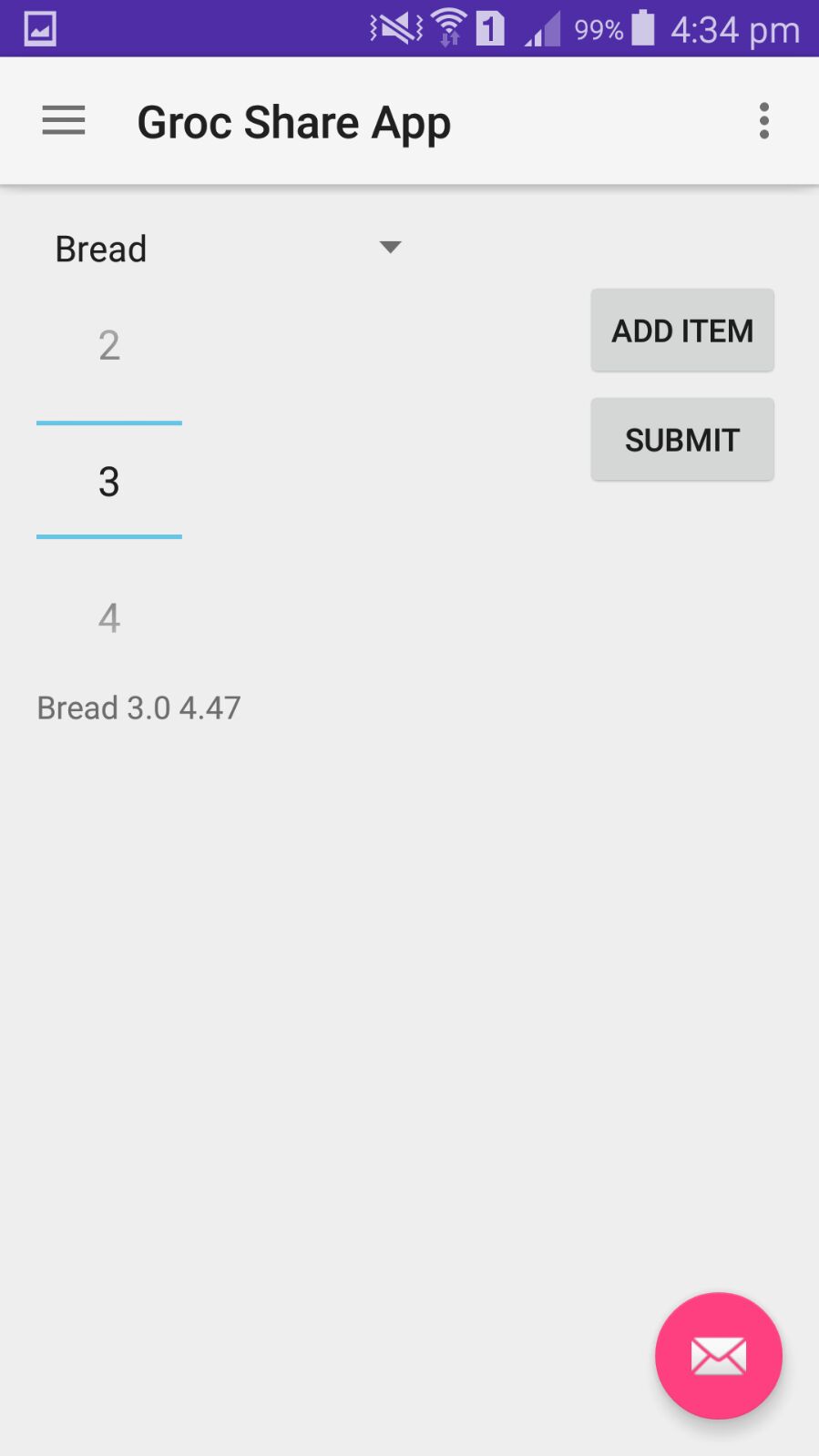


**Fig 3.1 : Sign In activity Fig 3.2 : Sign In Page**

***STEP 2.1: ORDER BASED PLACEMENT***

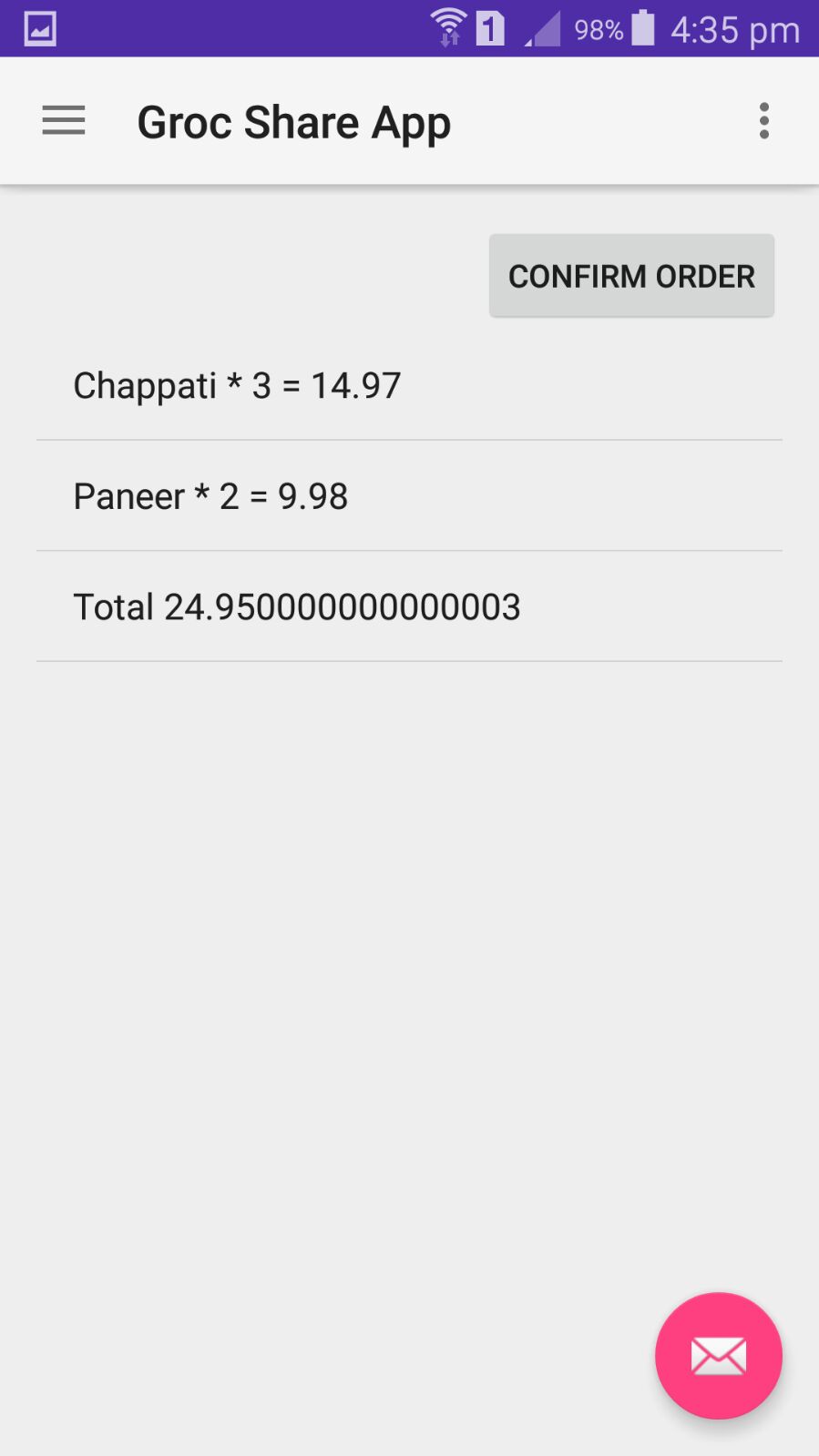


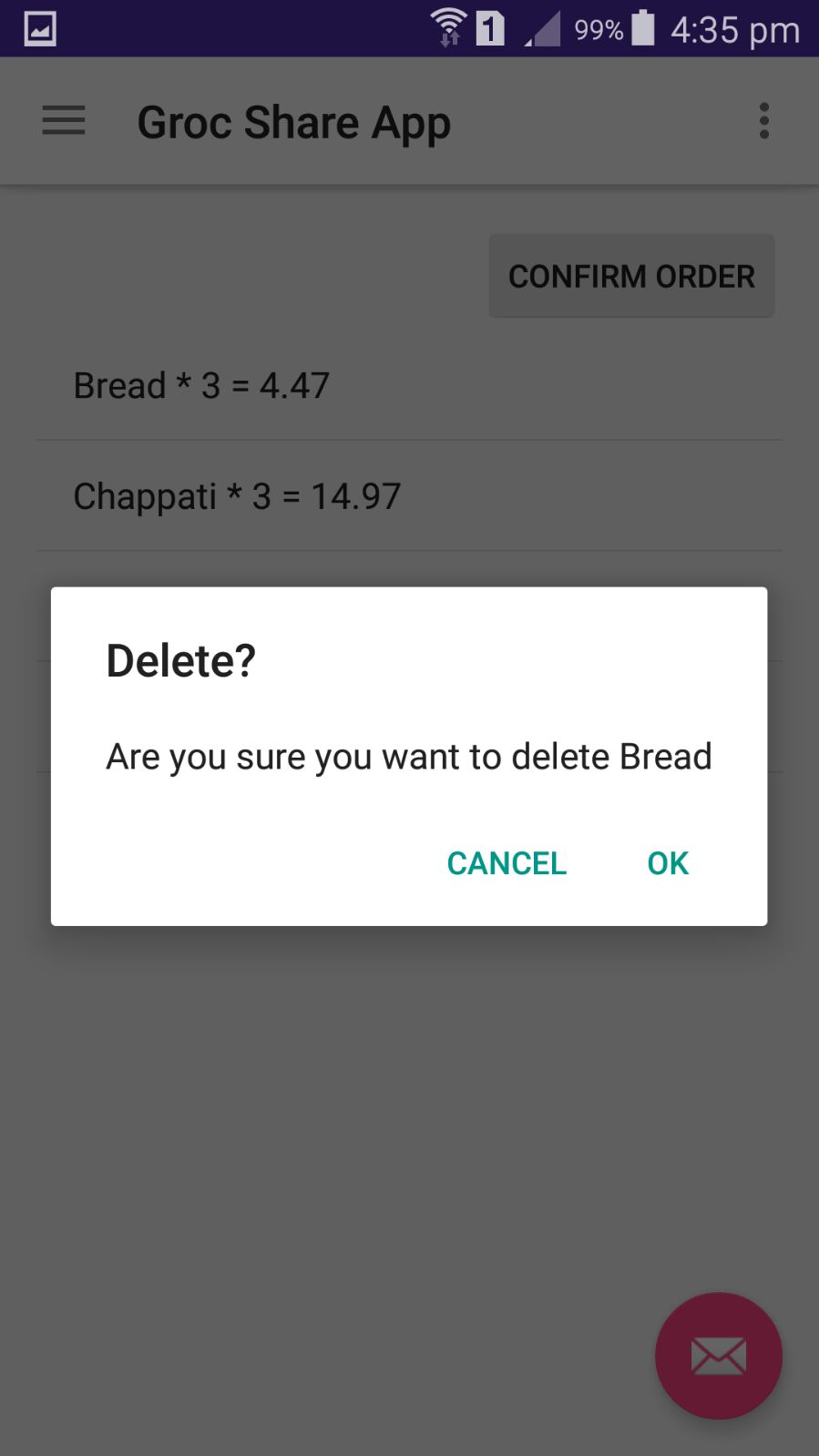
**Fig 4.1 : Order Screen Fig 4.1 : Menu Items**



**Fig 4.3 : Order Item Placed**

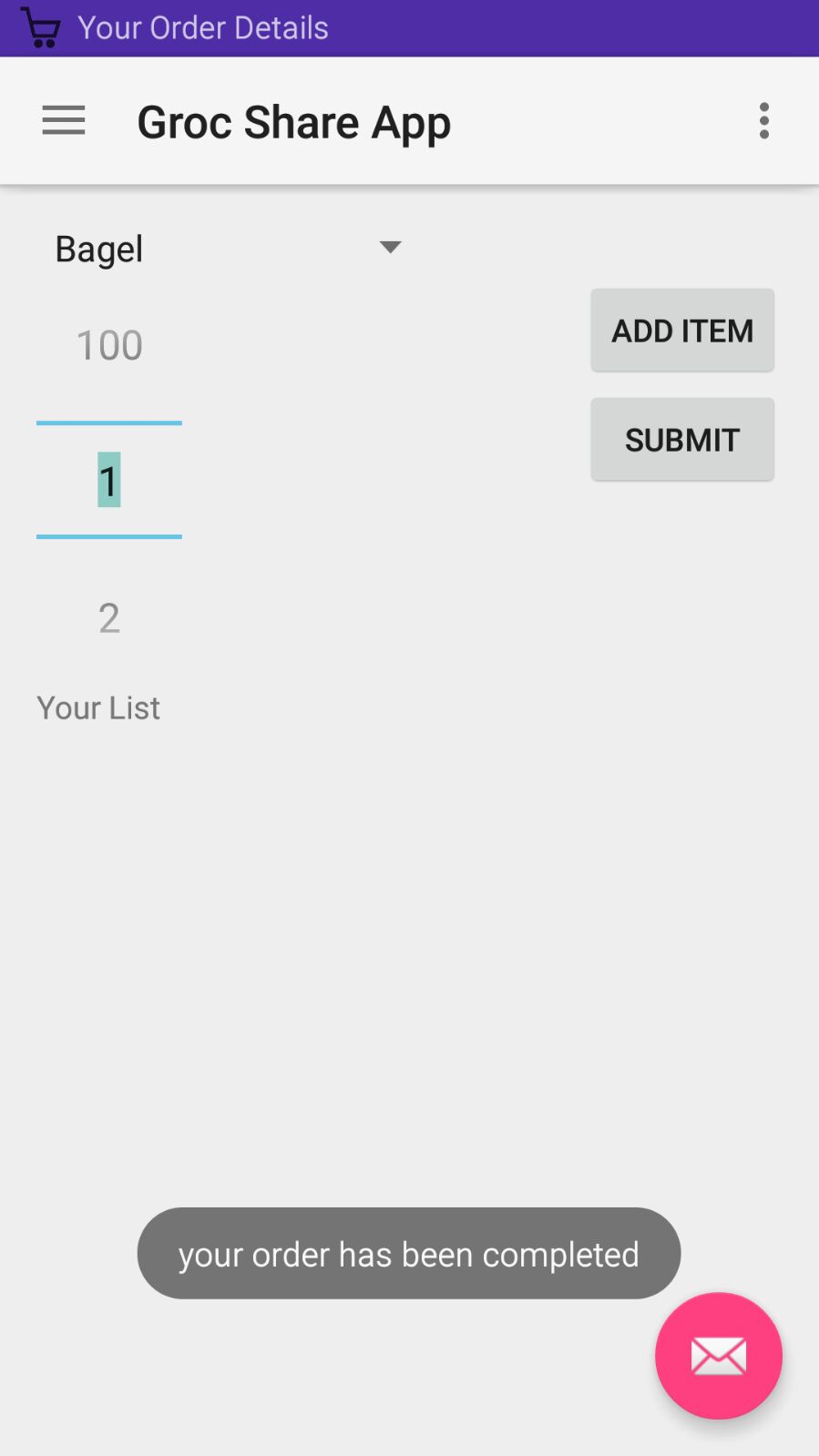
***STEP 2.2: ORDER BASED PLACEMENT***





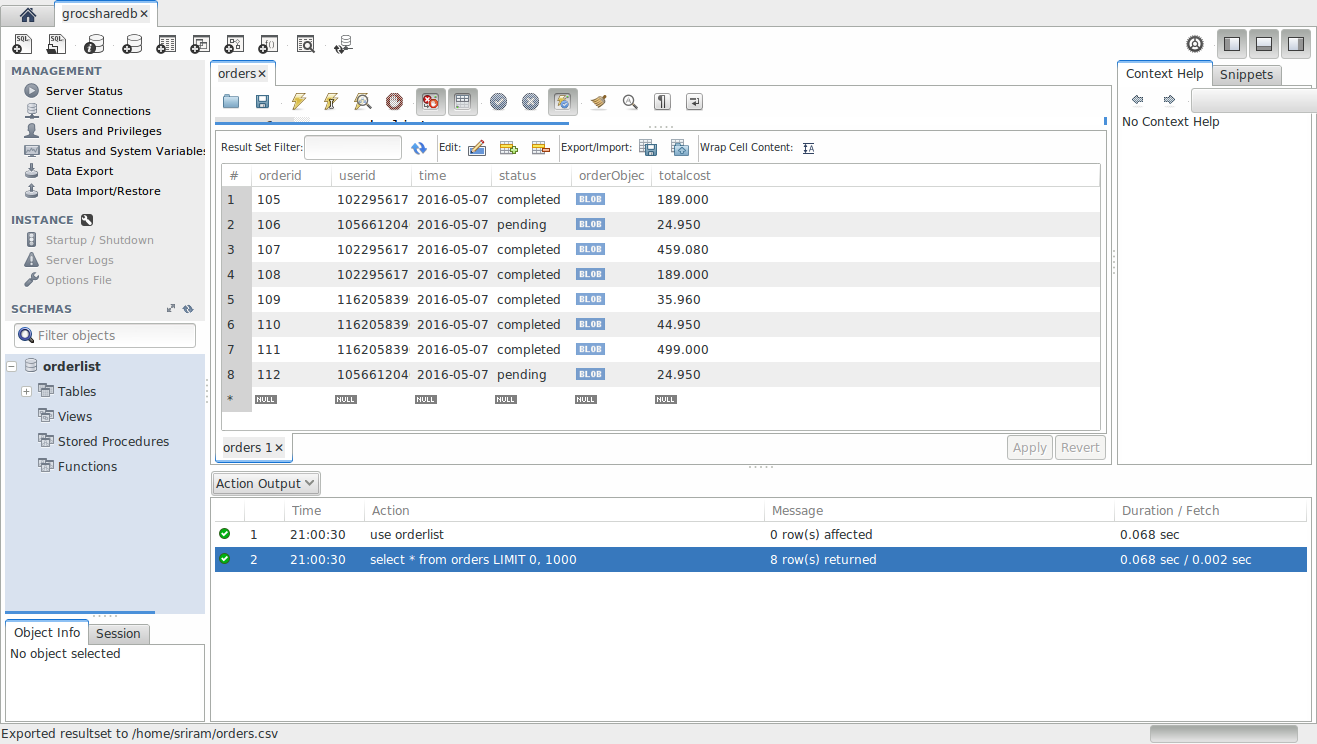
**Fig 4.4 : Delete Items Fig 4.5 : Final Order**

***STEP 2.3: ORDER CONFIRMATION***



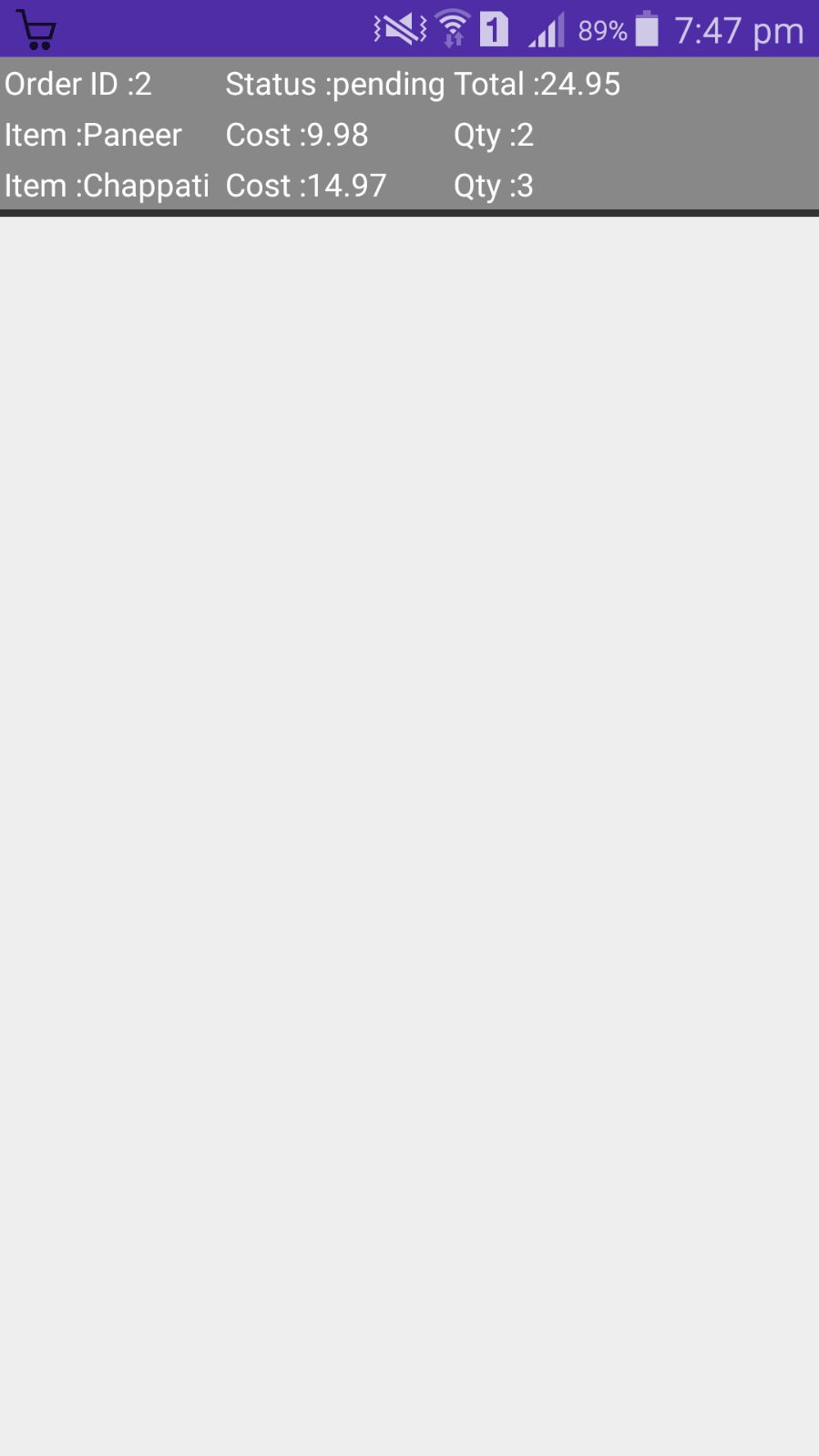
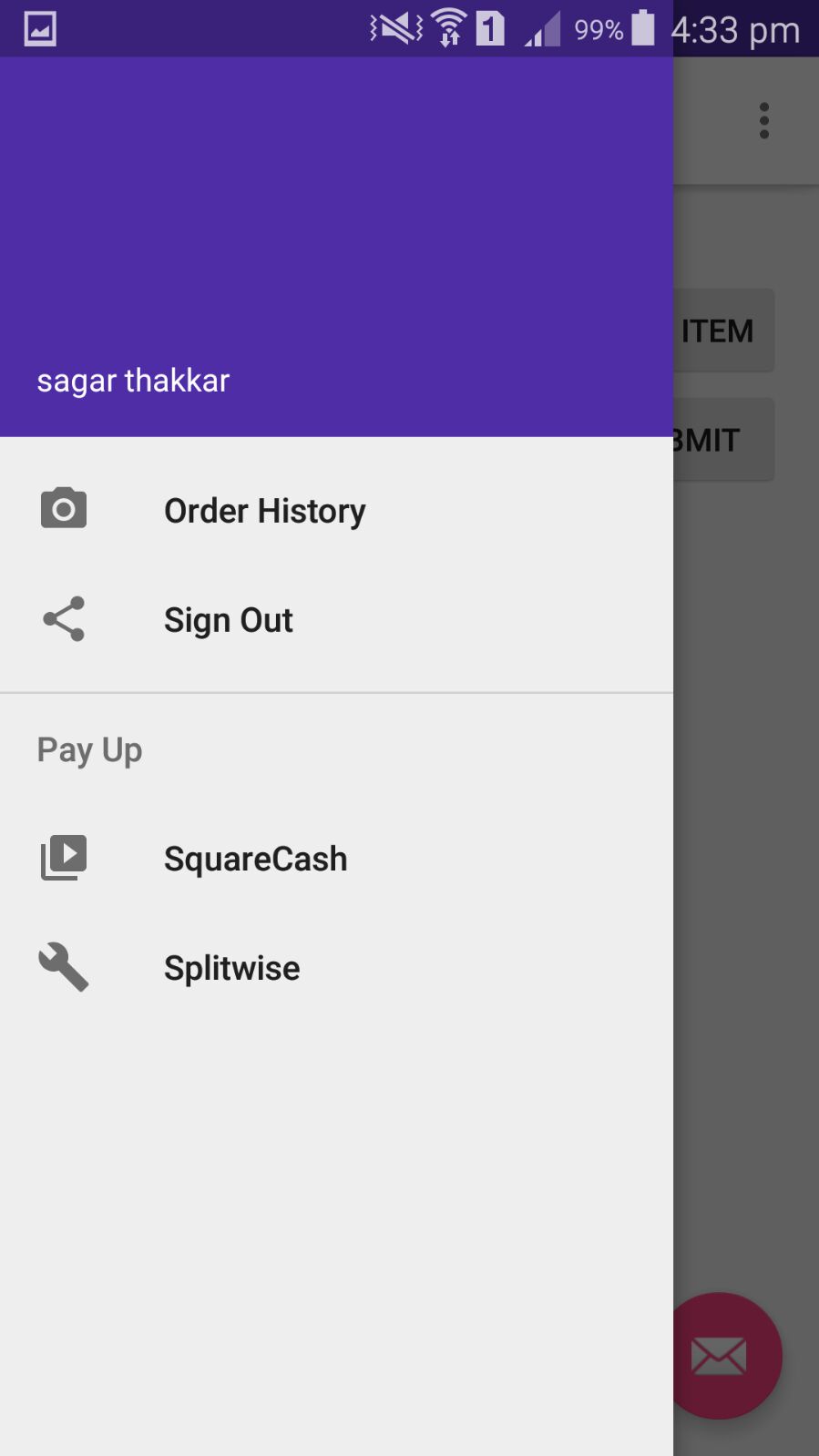
**Fig 4.6 : Final Order Confirmation**

***STEP 3: CLOUD DATABASE***



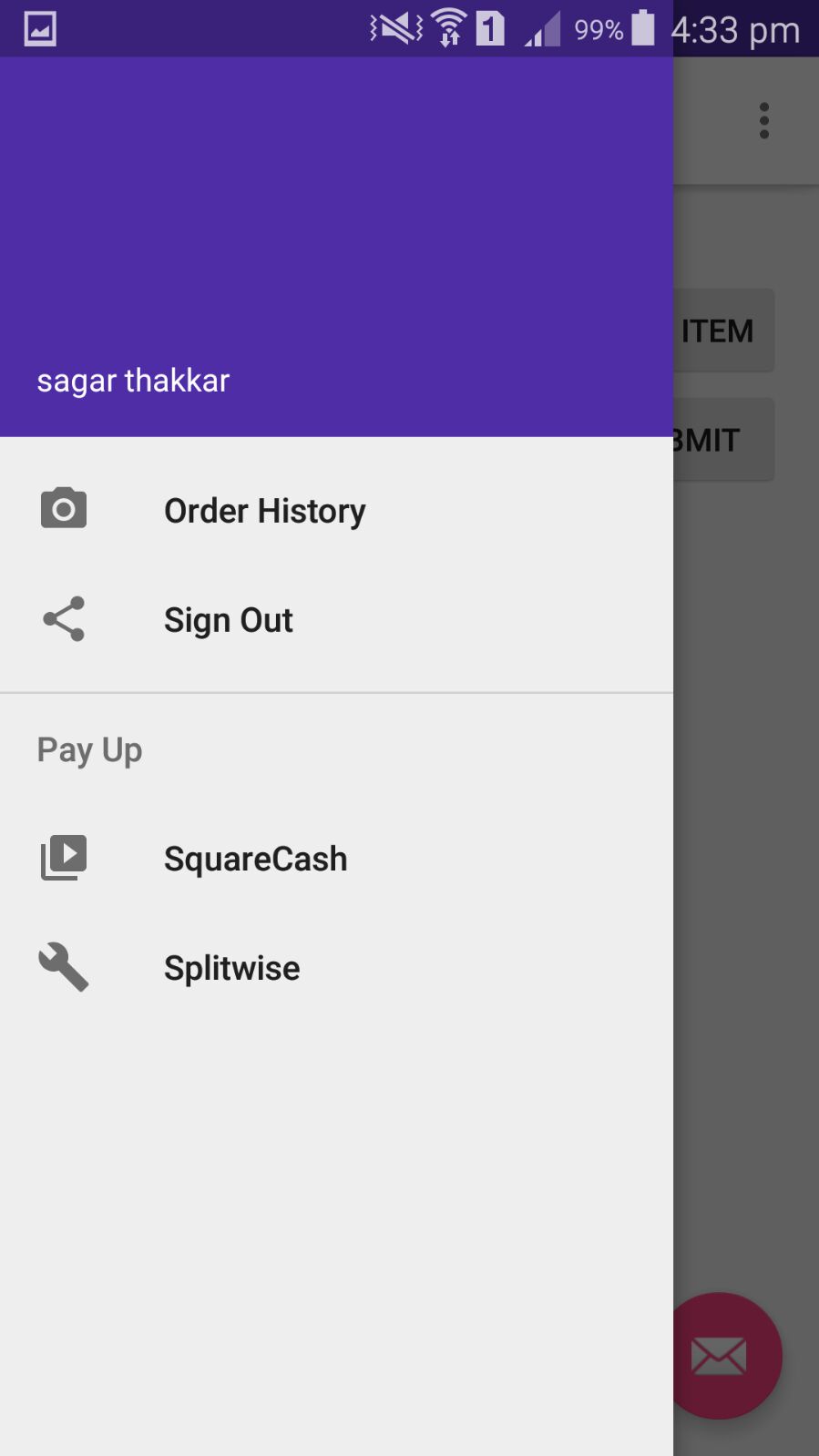
**Fig 5 :Database**

***STEP 4 : ORDER HISTORY***



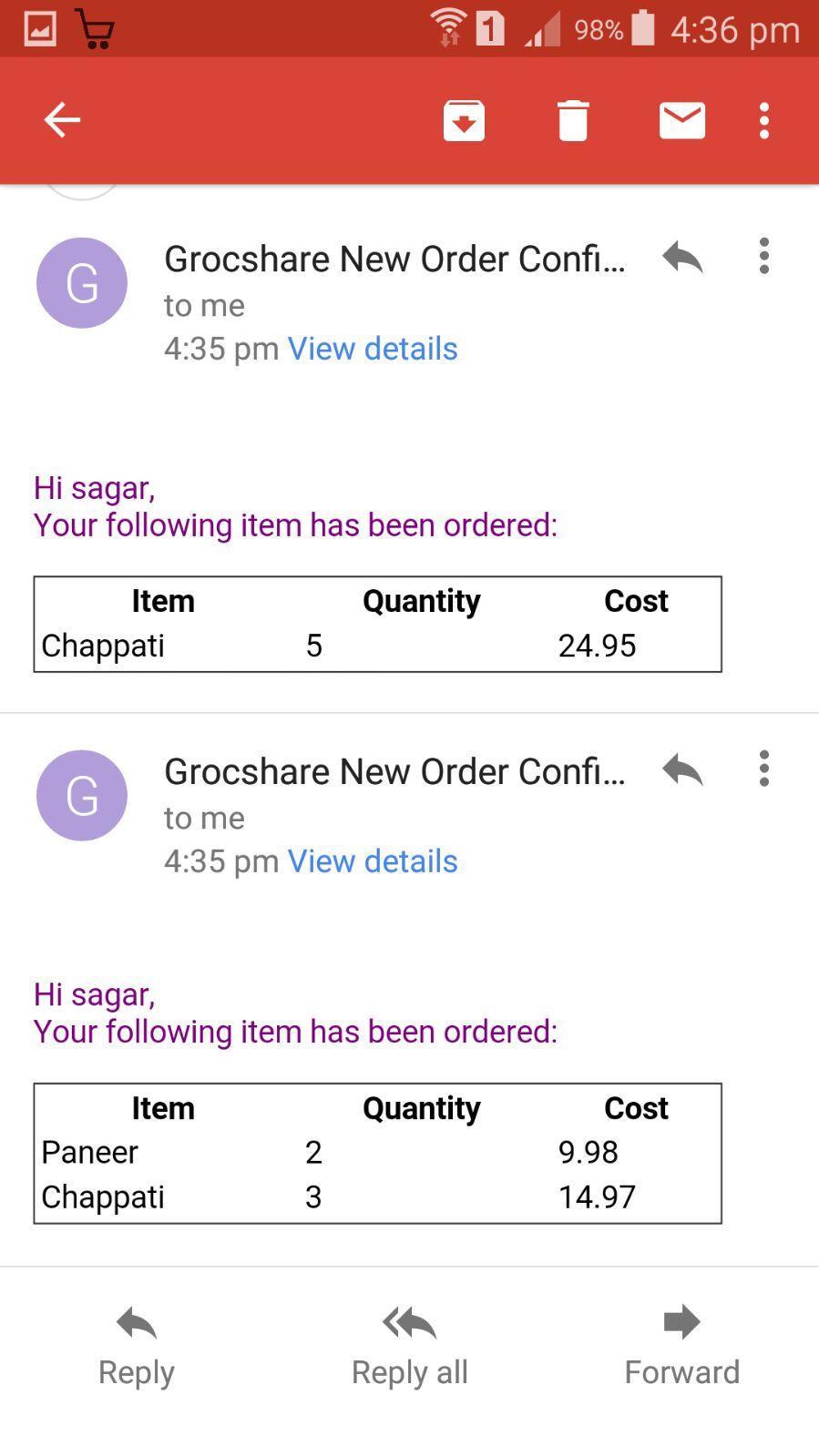
**Fig 6.1 :Order History Option 6.2 :Order History Item**

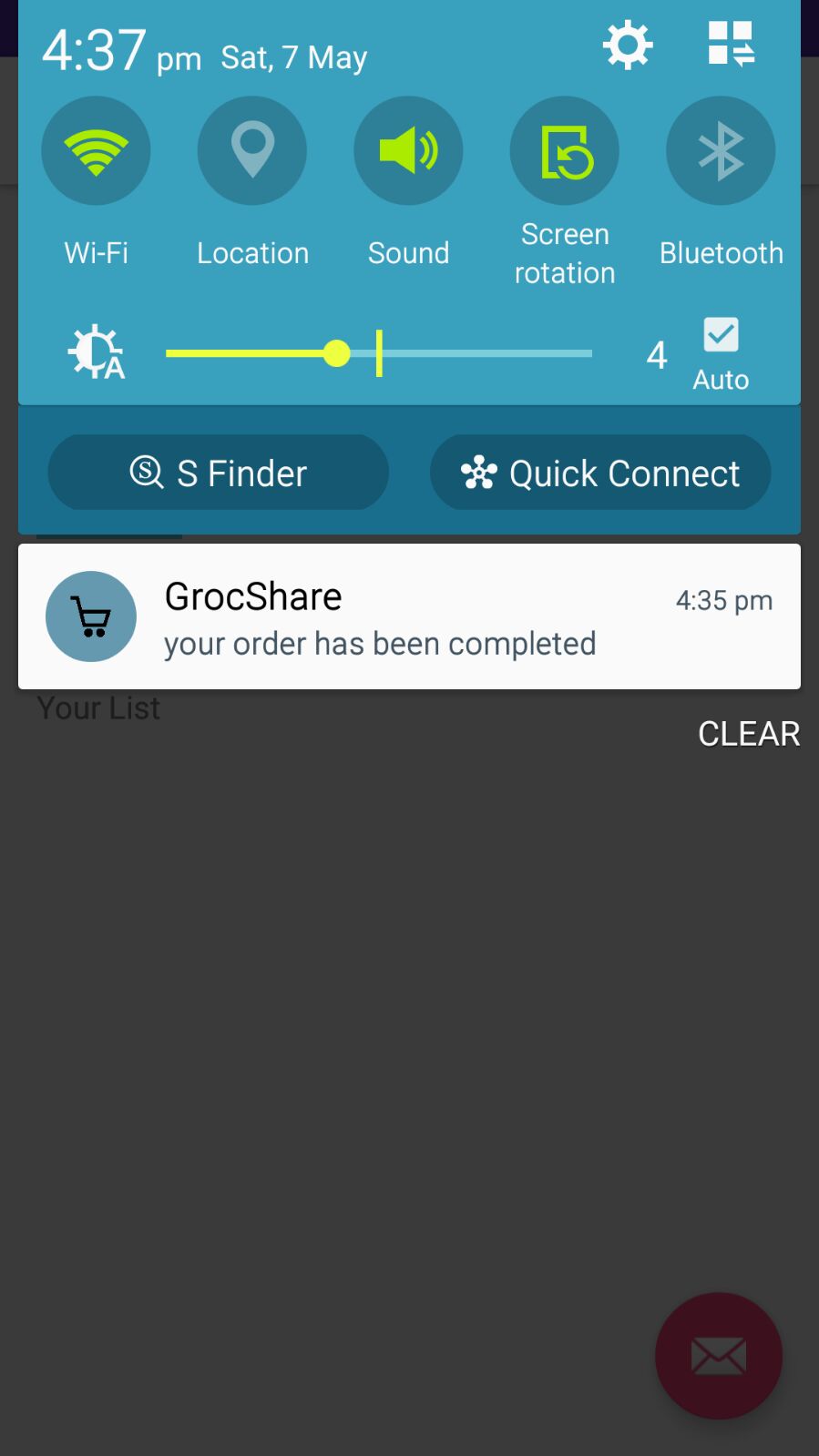
***STEP 5: UTILITY APPS***



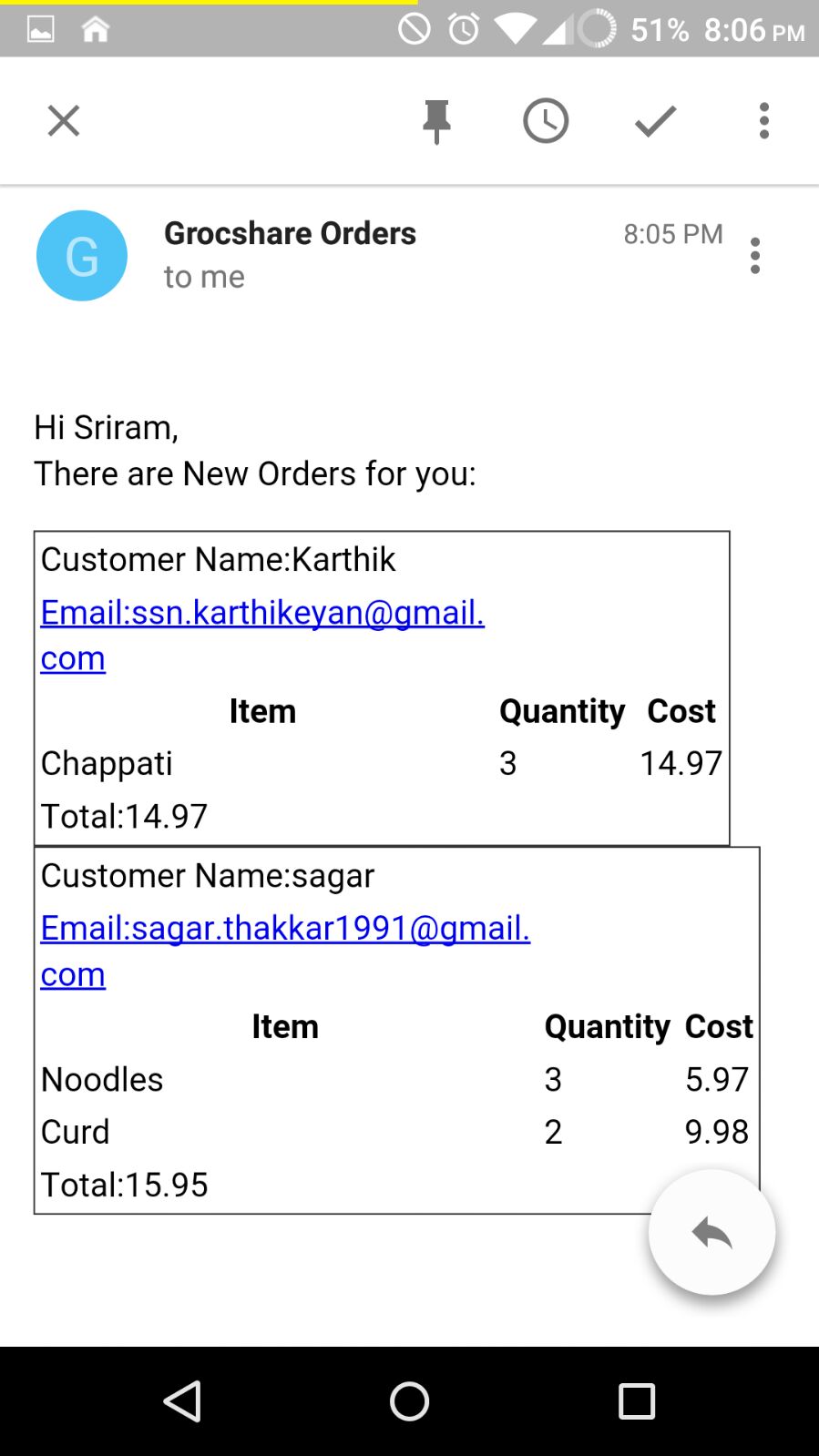
**Fig 7 :Utility App Options**

***STEP :6 ORDER NOTIFICATIONS***





**Fig 8.1:GCM Notification Fig 8.2 :GMAIL Notification**



**Fig 8.3:Vendor Notification**

**Pros and Challenges:**

1. Solves real time problem of travelling to stores to purchase grocery.
2. Help to leverage the benefit of order pooling. Can save some money.
3. Reduced problems of grocery during emergencies.
4. Post order analysis of data can be a boon for the business.
5. Integration of store data and dynamic prices of the items in the grocery stores shall be available before hand.

**Possible Advances**

With Current design it is only helpful to users in a particular geographical area. An advanced version of the app shall do location based merging which will facilitate both the users and vendors.Keeping the track of orders sent we can analyse the data to understand the pattern of purchase,average purchase amount, total order till date etc which on creation and presentation can benefit both the buyers and sellers to better concentrate on their demand and supply chain.

**Conclusion**

The android application developed gave us thorough understanding of the activities, processes and cloud integration. The current version is ready to use and over the course can develop the functionalities and can probably shape up in a revolutionary entrepreneurial idea. This app thus will solve real life problem and make the grocery purchase a better and and pleasant experience.

**References:**

<https://github.com/gabrielemariotti/cardslib/blob/master/doc/GUIDE.md>

<http://stackoverflow.com/questions/7181534/http-post-using-json-in-java>

<https://github.com/codepath/android_guides/wiki/Fragment-Navigation-Drawer>

<https://developers.google.com/identity/sign-in/web/backend-auth>

<http://www.riskcompletefailure.com/2013/11/client-server-authentication-with-id.html> <http://www.riskcompletefailure.com/2013/11/client-server-authentication-with-id.html>

<http://android-developers.blogspot.in/2013/01/verifying-back-end-calls-from-android.html>

<http://www.riskcompletefailure.com/2013/11/client-server-authentication-with-id.html>

<https://github.com/googlesamples/google-services/blob/master/android/signin/app/src/main/java/com/google/samples/quickstart/signin/IdTokenActivity.java>