50.001 Introduction to Information Systems and Programming

1D Android Application Project (Team 3-1)

sentiENTERPRIZE

Anirudh Shrinivason Gerald Hoo James R Tiovalen Jia Shuyi V S Ragul Balaji

Background

Point-of-Sales (POS) systems serve a crucial purpose in several SME applications. Currently, most POS solutions have disjointed points-of-sale and manual inventory management[1]. Furthermore, they are also very costly[2]. These POS systems also utilize third-party services which are relatively expensive. The current plight of POS systems in the market forces SMEs to rely on such expensive solutions without many viable alternatives and remains an avenue for wasted expenditure on unnecessary features in many cases. The constraints and needs of business owners are highlighted in the following table:

Constraints	Needs
Tight Margins	Accessibility
High Fees	Simplicity
Time as Externality	Ease of Use

Solution

sentiENTERPRIZE addresses the aforementioned problems by centralizing the point-of-sale and automating inventory management onto Android devices, thereby severely reducing infrastructure costs and increasing accessibility of our solution. This app intends to replace the overpriced existing ERP (Enterprise Resource Planning) systems that the majority of SMEs cannot afford. Our compact yet powerful solution does away the unnecessary complexities involved in resource planning, and lets users focus on the business needs and strategies.

Features

- **Point of Sales**: Point of sales allows the user to delve deep into the sales history and track transaction details. It provides a venue for users to add transactions, and record sales accurately in real time.
- **Inventory Management**: Inventory management offers a bird's eye view of the current inventory, in which one can easily add, edit or remove entries. Users can also post live comments within each entry, which allows for cross-department colleagues to quickly communicate if any issues were to occur. In addition, one can also star customers' favourite items under the "Favourite" tab.
- **Profile Page (Overview of Statistics)**: Profile statistics provides a quick summary of the business over a period of time to accurately contribute insights to the user.

[1] *8 Common POS System Problems*. https://accucode.com/8-common-pos-system-problems-how-to-solve-them/

[2] *How Much Does A POS System Cost In Singapore? A Price Guide*, Tashya Viknesh. https://blog.thunderquote.com/2017/09/06/how-much-does-a-pos-system-cost-in-singapore-a-price-guide-thunderquote/

Innovation and Design

Our mobile app implementation of POS systems is a convenient and cheap alternative to conduct resource tracking and inventory management. Mobile apps allow an easier venue to update features, incorporate timely bug fixes and push systematic version releases. Big data and machine learning can also be utilized in alignment with the collected data to run demand prediction. Our UI design has also been customised for ease of use and is suited to members of all age groups thereby mitigating any technological barriers from the user's perspective. Moreover, our solution of running it on Android devices allows for the user to worry less regarding the hardware security of the POS system as it is (often) sufficiently ensured by the Android device manufacturer.

System Design and Implementation

System Architecture

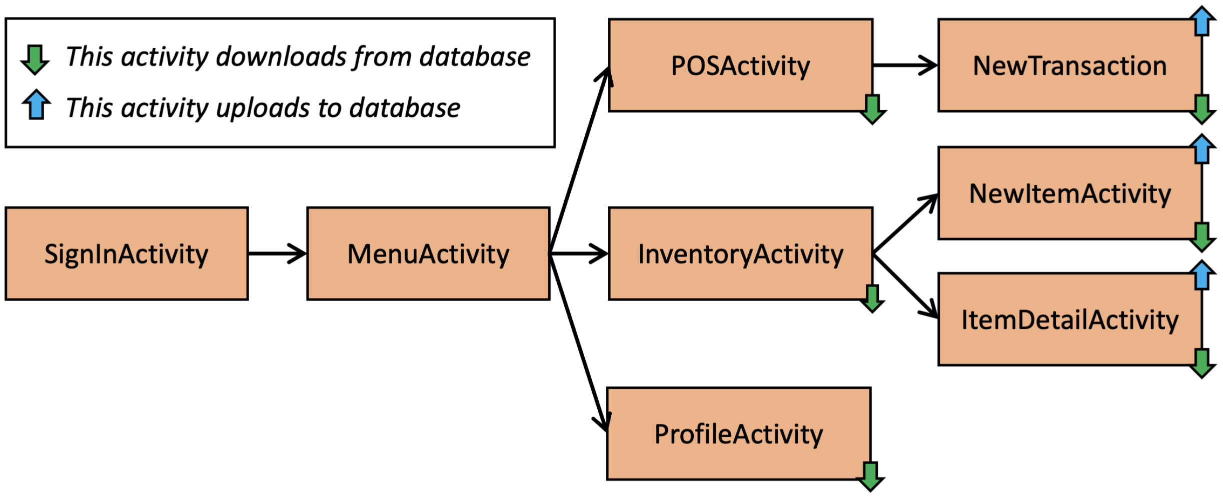
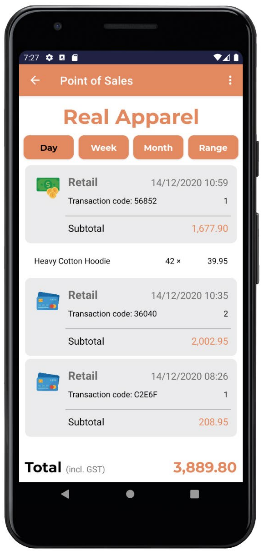


Figure 1: System Architecture – Application Activities Overview

App Screenshots


7:27

Point of Sales

Real Apparel

Day | Week | Month | Range

Retail 14/12/2020 10:59
Transaction code: 56852 1
Subtotal 1,677.90

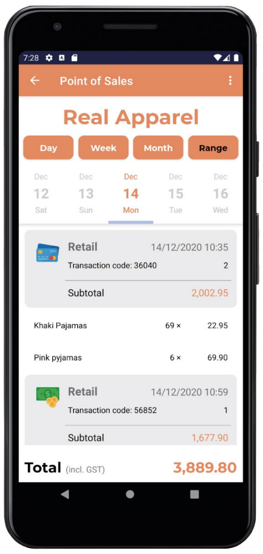
Heavy Cotton Hoodie 42 × 39.95

Retail 14/12/2020 10:35
Transaction code: 36040 2
Subtotal 2,002.95

Retail 14/12/2020 08:26
Transaction code: C2E6F 1
Subtotal 208.95

Total (incl. GST) 3,889.80

POS (Day)


7:28

Point of Sales

Real Apparel

Day | Week | Month | Range

Dec 12 Sat | Dec 13 Sun | Dec 14 Mon | Dec 15 Tue | Dec 16 Wed

Retail 14/12/2020 10:35
Transaction code: 36040 2
Subtotal 2,002.95

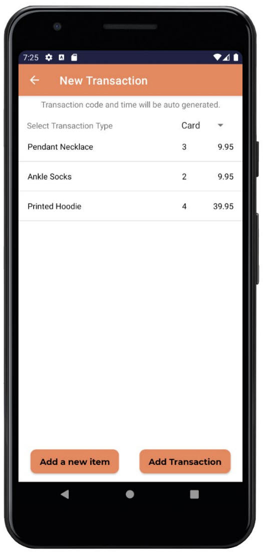
Khaki Pajamas 69 × 22.95

Pink pyjamas 6 × 69.90

Retail 14/12/2020 10:59
Transaction code: 56852 1
Subtotal 1,677.90

Total (incl. GST) 3,889.80

POS (Range)


7:25

New Transaction

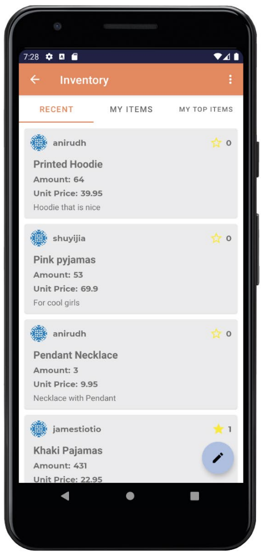
Transaction code and time will be auto generated.

Select Transaction Type Card

Pendant Necklace	3	9.95
Ankle Socks	2	9.95
Printed Hoodie	4	39.95

Add a new item | Add Transaction

POS (New Transaction)


7:28

Inventory

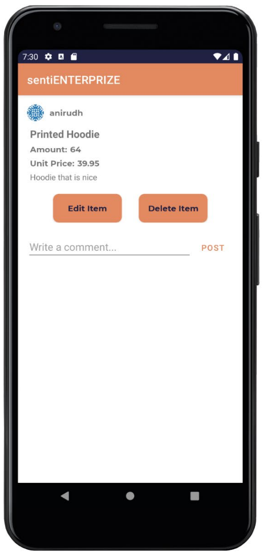
RECENT | MY ITEMS | MY TOP ITEMS

anirudh ☆ 0
Printed Hoodie
Amount: 64
Unit Price: 39.95
Hoodie that is nice

shuyijia ☆ 0
Pink pyjamas
Amount: 53
Unit Price: 69.9
For cool girls

anirudh ☆ 0
Pendant Necklace
Amount: 3
Unit Price: 9.95
Necklace with Pendant

jamestiotio ★ 1
Khaki Pajamas
Amount: 431
Unit Price: 22.95

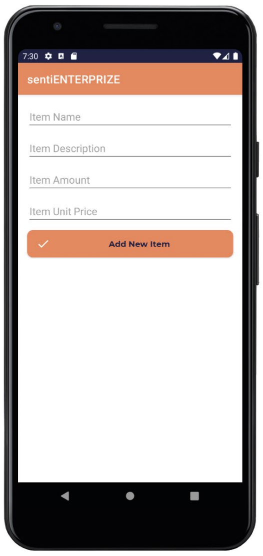

7:30

sentiENTERPRIZE

anirudh
Printed Hoodie
Amount: 64
Unit Price: 39.95
Hoodie that is nice

Edit Item | Delete Item

Write a comment... POST


7:30

sentiENTERPRIZE

Item Name

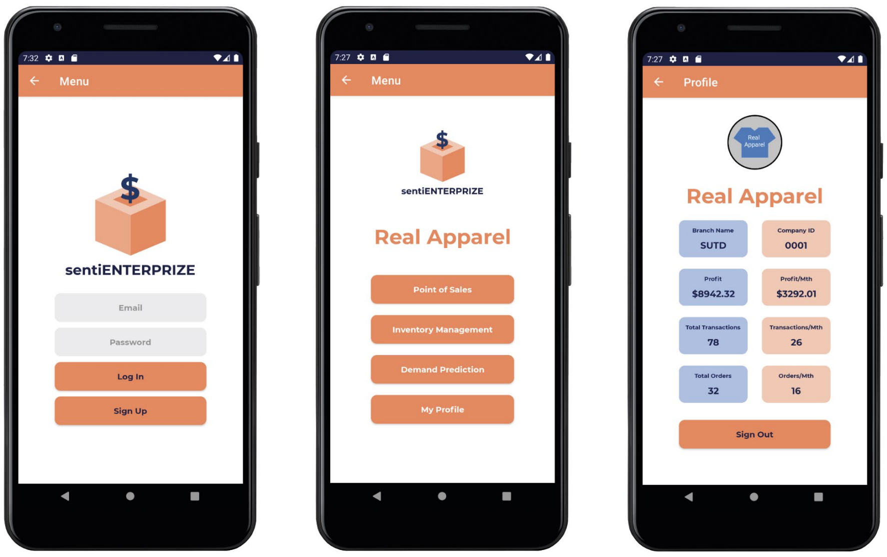
Item Description

Item Amount

Item Unit Price

Add New Item

Inventory Management (Inventory, Item Comments, New Item)



Branch Name SUTD	Company ID 0001
Profit $8942.32	Profit/Mth $3292.01
Total Transactions 78	Transactions/Mth 26
Total Orders 32	Orders/Mth 16

Login Page Menu Page Profile Page

Figure 2: Overview of the various pages of our application.

Design Patterns

Firebase – Observer Design Pattern

Our system is connected to the Firebase Authentication & Database System in an event-based mechanism. We implemented event listeners which follow the observer design pattern. The event listener is implemented by inheriting *onDataChange()* from Firebase's *ValueEventListener* class. If there are any changes that happen in the Firebase - either the user being logged out, their credentials being deleted from the database, a new follow-up thread message on an item's detail page or an update to an item's amount and unit price - everything will be updated in real-time on all of the devices currently connected to the sentiENTERPRIZE backend. The user does not need to keep refreshing the activity or restarting the app to see new changes being implemented. This greatly increases productivity and convenience for the company that uses sentiENTERPRIZE.

Data-Passing between *DialogueFragment* and *Activity* - Modified Observer Design Pattern

To pass data effectively from the *DialogueFragment* to its calling Activity *NewTransaction*, we implemented a modified observer design pattern. Our *DialogueFragment* acts as the subject, and it has one and only one subscriber – *NewTransaction*. The interface *OnCompleteListener* implemented in *DialogueFragment* has an abstract method named *onComplete*, which is analogous to the *notifyObservers()* method in a normal observer design pattern. When *onComplete* is invoked (by pressing a button), the data is updated in the observer, *NewTransaction*. Thus, a list of items waiting to be added can be shown.

Adapter Design Pattern

For both our comment functionality under *ItemDetailActivity* and our *ItemListFragment* (Fragment used to list our inventory items), we implemented an adapter design pattern. The *CommentAdapter* class inherits from the *RecyclerView.Adapter* class, which provides a binding from the app-specific data set to the corresponding views, and it is used to display the thread-based commenting system under each item's detail page. The *ItemListFragment* contains a reference to a *FirebaseRecyclerAdapter* object, which is used to set up a Query system connected to the Firebase database backend to check whether an item has been given a star or not.

Generics

We utilized generics for the *FirebaseRecyclerAdapter* object under *ItemListFragment*, so that it would only take in our custom-defined *Item* and *ItemViewHolder*-type objects. This is to ensure that only actual real items are being processed under the respective viewholders, and not transactions.

Additionally, our *CommentAdapter* which inherits from the *RecyclerView.Adapter* class also implements a generics-based check such that it would serve as a container for *CommentViewHolder* objects. This is to organize the different types of viewholders that we utilize in our application. As such, if we were to erroneously specify a different type of object under those aforementioned adapters, we would be able to catch any potential errors at compile-time.

Encapsulation & Inheritance

For the sake of best coding practices, we encapsulate different types of objects as models. We have specified several models, namely: User, Item and Comment for the Inventory Management page, as well as a Serializable Transaction Item for the POS page. These objects are encapsulated within their own classes with their respective attributes, data fields and methods, so that we do not have to re-define an item whenever we need to use an item.

We also implemented inheritance for both our POS Fragments and our Inventory Management Fragments. For our POS, we have *DayFragment*, *WeekFragment*, *MonthFragment* and *RangeFragment*, while for our inventory management, we have *RecentItemsFragment*, *MyItemsFragment* and *MyTopItemsFragment*. All of the aforementioned fragments inherit the base *Fragment* class.

Android UI Views & Layouts

Our application utilizes *ConstraintLayout* for the POS fragments, *LinearLayout* for each transaction listed in the POS page, as well as a combination of *RelativeLayout* and *LinearLayout* for our inventory management activities.

For our inventory management feature, we utilized Fragments to organize the three options of organizing the items: Recent Items, My Items and My Top Items. Each Fragment contains a *RecyclerView* to organize said items in the selected sorting system. The *RecyclerView* has an infinite scrolling mechanism, which allows the user to continue scrolling if they have numerous items located in the database.

Additionally, a user login persists across app restarts or orientation changes. This is because we utilized Firebase's persistent authentication feature, which is of a similar notion to *SharedPreferences*.

For each user's profile picture in the inventory management page, we implemented a background downloader task using *AsyncTask*. Currently, for demonstration purposes, we grabbed the respective profile pictures associated with the user's email from Gravatar. For future development, we could always change and adapt this accordingly to fetch either a company branch's ID, supervisor's name or manager's profile picture, depending on the company's needs.

Team Contributions

Name	Contributions
Anirudh Shrinivason	- Point-of-Sales Functions (Including Add Transactions, Update Inventory) - Firebase Integration between Transaction and Inventory management - Integration and Merge of Different Branches - Short Description of Project
Hoo Yong Wei, Gerald	- Overall UI and Logo (Figma) - Login Page, Menu Page, Profile Page - Standardized colors.xml and styles.xml - Integration and Merge of Different Branches - Poster Design
James Raphael Tiovalen	- Inventory Management Page (with Multiple Fragments: Main Page, Item Details Page, & Add Item Page) - Firebase Realtime Database Configuration Setup - Integration & Organization of Items, Users, Comments & Transactions with Firebase
Jia Shuyi	- Point-of-Sales Functions (Including Multiple fragments to Display Recent Transactions, Add Transactions, Update Inventory) - Firebase Integration between Transaction and Inventory Management - Integration and Merge of Different Branches - Short Description of Project
Velusamy Sathiakumar Ragul Balaji	- Session Management - Firebase Auth - Update Inventory Items Activity - Refactoring codebase - Exhibition Video

Table 1: List of contributions done by each of our team members.

Possible Future Work

Demand Prediction

Originally planned to be included in the release of our app were machine learning and statistical models to predict demand. These models can be used to alert the business if inventory would be too low to service immediate future demand. It could also be used to optimise the production by making the process just-in-time JIT, saving costs on warehousing and transportation.

One approach we would like to take for demand prediction was to use Random Forest Regressors (RFR) as an initial modelling method, training the model on a desktop using tools like scikit-learn and exporting the model as a PMML[3]. Then, on the Android device, we can run inference on the dataset using JPMML[4]. Using this model, we can input metadata about the day, time of day, branch and the model will attempt to use its historical data to predict demand for the day. The model would also cross-reference prediction data with inventory data to alert the user if the stock may run low soon, hence prompting the user to order more stock.

Sensor-Based Inventory Management

With the advent of IoT and decentralized supply chains, the number of warehouses and storage facilities of companies that adopt the usage of IoT technologies have been increasing at a commensurate rate. For future work, we would like to integrate this into our inventory management system by providing additional utilities such as barcode and QR scanner, RFID reader, and NFC tagging, all from a single supporting mobile phone (equipped with the corresponding sensors such as a camera, an RFID sensor and an NFC sensor), eliminating the requirement of purchasing any external hardware-based readers or scanners.

Automated POS System

Another avenue for possible work in future is the POS system. Currently, we provide means for the user to record down transactions manually in a simple manner. However, we would like to automate this process, and provide a solution to display and jot the transactions based on the scanning of a QR code or the swiping of a card based on the SME needs.

[3] *Predictive Model Markup Language.* https://en.wikipedia.org/wiki/Predictive_Model_Markup_Language

[4] *Java Class Model API for Predictive Model Markup Language (PMML).* https://github.com/jpmml/jpmml-model

Custom Search & Sort Functionality

An additional feature that would be good to have is a custom searching and sorting functionality for both items and transactions. This would provide a much faster access to a specific item or transaction for the user should the need ever arise.

Conclusion

In conclusion, sentiENTERPRIZE has a wide range of features that allows businesses to worry less on managing inventory and transactions and focus more on what matters for them. More importantly, our application uses various best practices for its various functionality implementations, and we developed it to be as modularized as possible. This allows for future features to be easily implemented into our future updates without much disruption to the businesses using our application. With these practices, it also allows for our code to run more efficiently, allowing businesses to have a quick and worry-free experience on our application. As such, companies would be able to have a much smoother time in managing their businesses with significantly less cost. Truly, a positive and powerful enterprise for all.

References

8 Common POS System Problems. Retrieved from https://accucode.com/8-common-pos-system-problems-how-to-solve-them/

How Much Does A POS System Cost In Singapore? A Price Guide, Tashya Viknesh. Retrieved from https://blog.thunderquote.com/2017/09/06/how-much-does-a-pos-system-cost-in-singapore-a-price-guide-thunderquote/

Java Class Model API for Predictive Model Markup Language (PMML). Retrieved from https://github.com/jpmml/jpmml-model

Predictive Model Markup Language. Retrieved from https://en.wikipedia.org/wiki/Predictive_Model_Markup_Language