Deliverable K: User and Product Manual

Deliverable K: User and Product Manual

Engineering Design

GNG 1103-A

Team Members:

- Hatim Shakir.
- Rohit Sharma.
- Nour Elali.
- Farah El Siss.
- Michael Abu El Farag.

Deliverable due date: December 8, 2021.

Professor: Dr. David Knox.

TAs & PM: David Nku, Pankaj Rathi, & Elsa Lange.

Faculty of Engineering

Fall 2021

Table of Contents

List of Acronyms and Glossary

Table 1. Acronyms

Acronym	Definition
UPM	User and Product Manual
APK	Android Application Package
UI/UX	User Interface/User experience
ID	Identity Document
API	Application Programming Interface
FAQ	Frequently Asked Questions
OS	Operating System

Table 2. Glossary

Term	Definition
Node-Red	Node-RED is a flow-based development tool for visual programming developed originally by IBM.
MonogDB	MongoDB is a source-available cross-platform document-oriented database program.
Figma	Figma is a vector graphics editor and prototyping tool which is primarily web-based, with additional offline features enabled by desktop applications for macOS and Windows.
Flutter	Flutter is an open-source UI software development kit created by Google.

Term	Definition
Firebase	Firebase is a platform developed by Google for creating mobile and web applications.
Notion	Notion is a piece of notetaking software and project management software.
Wrike	Wrike, Inc is a project management application
LucidCharts	Lucidchart is a web-based proprietary platform that allows users to collaborate on drawing, revising and sharing charts and diagrams.

Introduction

Our primary goal is to develop a strong and effective loyalty platform that will allow various users (banks, clients, and retailers) to access and control their accounts while providing genuine value. In this user and product manual, we will be providing a deeper overview of our product and interpreting all the steps that were taken to build it. Moreover, this User and Product Manual (UPM) provides the information necessary for bank clients and retailers to effectively use our platform and for prototype documentation.

Overview

There are many problems with loyalty reward programs today. Users often struggle to use multiple rewards platforms and point systems, which have inefficient ways to save money, as they are hard to keep up with. Smaller retailers do not have the resources and the right management processes to start their own programs.

To democratize loyalty rewards we need to find a way to allow smaller retailers to thrive and make that process simple for the everyday bank client. A need exists to create a highly secure, simple, and compatible platform that can allow users to keep track of loyalty points, and their regular banking information for usage on various different retail stores, products, and services.

Our product fulfills the following needs. Bank clients have the ability to redeem and store points. 2-factor authentication for extra security. With different users, there needs to be a personalized experience on the platform. As a retailer, there should be the ability o set your own points, ads, and have full control over the management of their loyalty

rewards program. All users have easy navigation and browsing, as well as, be accessible from multiple devices.

Our platform is integrated with existing bank infrastructure, so it is already accessible to all users with certain banks that partner with Zaffin to use this product.

We are different from platforms today as we give the power pack to all retailers no matter their size. Allowing all retailers set their own loyalty rewards program by setting their own point per dollar and attract their customers by setting ads with target audiences. This platform gives them the resources to do so.

The everyday bank client has access to personalized offers depending on their purchasing history, as well as a dashboard with all the eligible retailers they can spend their points at. They can spend specific amount of points at specific retailers.

The platform's UI is built on Figma and functionality using MongoDB and NodeRED.

This product truly democratizes loyalty rewards by allowing retailers to have full control over their program and providing them with resources to do well. Bank clients can view all their rewards from one place, and are able to have full control over how they spend/ transfer their points.

All parties, such as bank clients, banks, and retailers, make or save money.

Cautions and Warnings

Although our platform is secure and requires a two-factor authentication system, we still ask users to not include personal information such as personal ID numbers, vehicle ID, etc..

Getting Started

In order to understand this product manual, we have presented our findings in a logical manner starting with the identification of user needs. Once the problem was identified, we gathered and prioritized user needs. In order to adhere to client needs, we made sure that our application would accomplish the standard for current loyalty rewards systems such as PC Optimum. Hence, our program was designed to take important client considerations

After this, we have carefully highlighted the different resources we used in a Bill of Materials table and associated the cost required for each component of this project. We then developed our 3 prototypes tackling front and back-end aspects of the application. We finally arrived at a finished project which was then tested to see that all critical aspects have been identified and all requirements have been met.

Set-up Considerations

In order to recreate this project, it is important to understand the different applications we used. To build our prototypes, we first used Flutter software. In order to get the system configured, we downloaded an additional Android Application Package (APK) which allowed us to write code into flutter and test it on an online android phone provided by the APK software.

In order to configure the node-red system, we downloaded node-red v.2.1.3 from https://nodered.org/ and then followed the instructions provided in the 'Get started' section. The last step was to run the node-red software in the local browser. This is achieved by opening the command prompt and simply typing 'node-red'. Once this is opened, a dashboard of nodes will be available where the program can be built in so-called flows.

To get a preview of the UI dashboard, the same localhost link with the added '/ui' extension can be added. This allows the user to view the final web page.

We also made use of the MongoDB database where we made a free account and stored information into different categories. This database was then connected to node-red by the use of an API. To create the API, we simply installed visual studio which is an IDE to write code.

User Access Considerations

Users are free to access any part of the platform, as long as they are verified and authorized to do so. This is achieved by validating the user login credentials, thus, allowing only registered and approved retailers and customers to log in and manage their points system and points respectively. While consumers are free to make as many accounts as they wish, retailers can only have one account that will be linked to their store addresses, and any attempt to make another account will be rejected and fail.

System Organization and Navigation

The UI design is as follows: the page presents the user with a login choice of a regular user or a retailer. Once picking one of these choices the user will have to login with a two-factor authentication. The regular users are presented with a dashboard that outlines offers, their total points, their points activity (points transactions history), and ways to redeem their points. Retailers are presented with a dashboard as well, however, with a different configuration and features. The following is displayed on a retailers dashboard: resource & analytics tab, adding different advertisements, points transaction history in their store(s), and rewards/item points management. To summarize, the general layout of the UI strictly avoids complicated routes and focuses on presenting information tabs on a dashboard which provides key details rather than cluttering the UI.

The UX design displays a merchant page and a user page. The merchant page and the user page both have a login page that can be accessed with proper credentials. Upon accessing the user page the user can enter a transaction number and gain points, meanwhile, for the merchant page the merchant can enter a new item and assign a point value to that particular item. The item added by the merchant will appear on a table that will catalog these items.

Exiting the System

Simply close the localhost, and Node-RED flow tabs and close the Node-RED terminal. These steps for exiting can be applied to exit the setup.

For exiting the system via the UI layout, the user or retailer will simply have to click their profile and click "log out", which will bring the users back to the login page.

What the final solution is

The following sub-sections provide detailed, instructions on what our solution is and how to use the various functions or features of our platform integrated onto a bank's existing platform, where users can access using a portal on their bank's app, and banks can manage just like any other product they have.

All Users Features

Security

Both retailers and bank clients have good security as they would have to use their regular bank log in to access the portal. After their login information is inputted into the system, we use a two-factor authentication, where a user would have to verify their login through a 3rd party app.

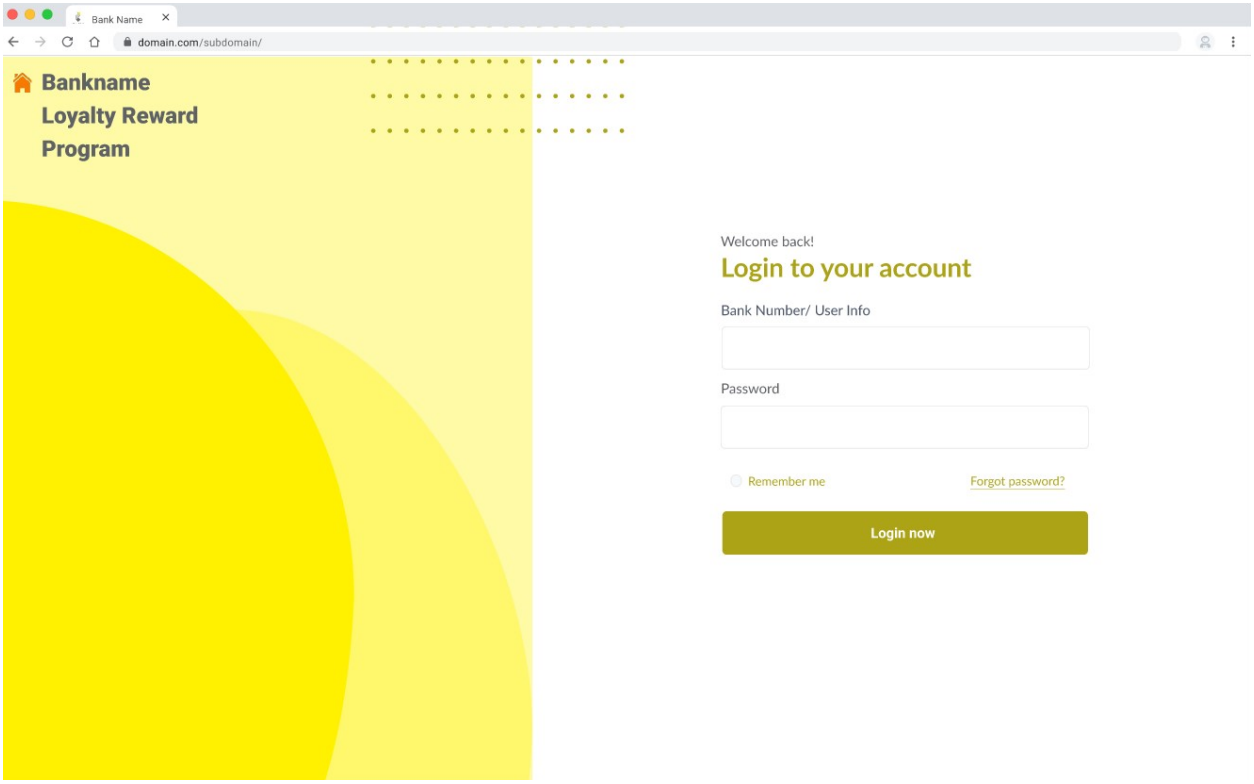

Figure 1. Interface for user to enter their login information with the bank.

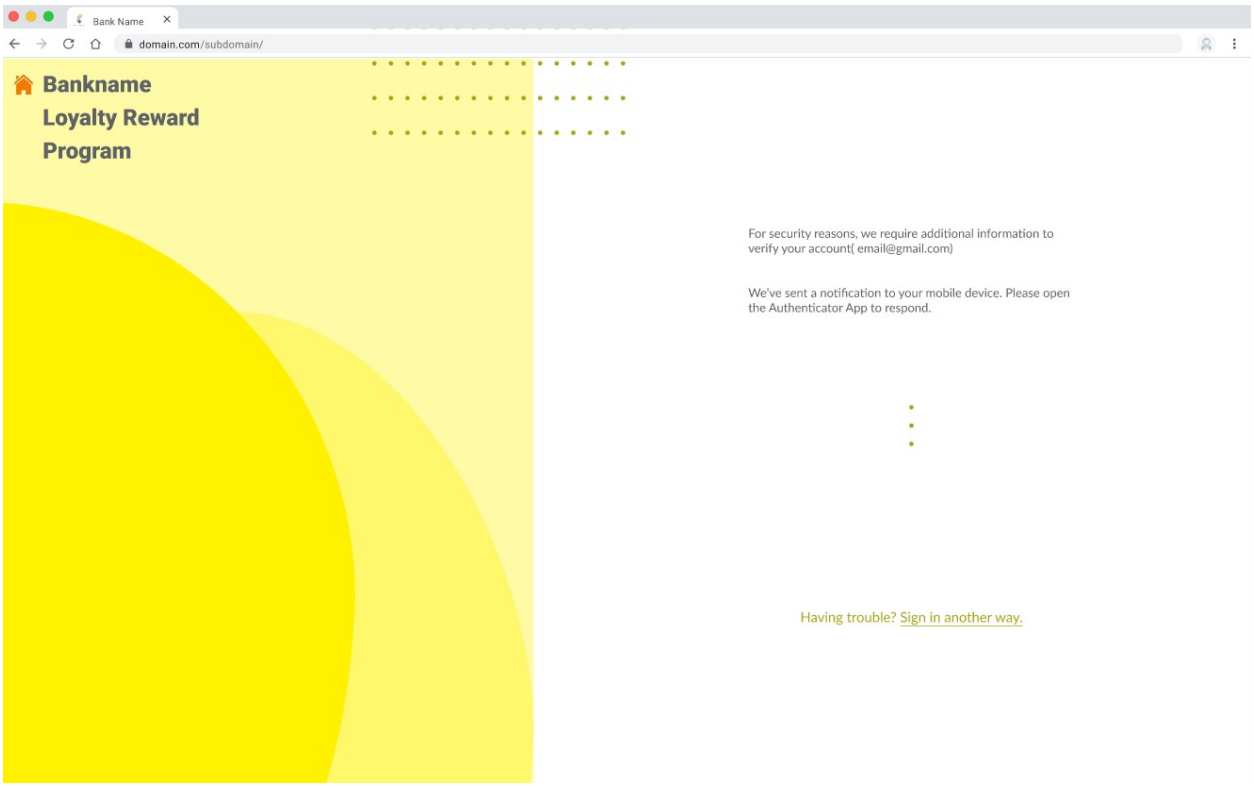

Figure 2. Verification Loading Page- Interface where user waits for verification being sent.

Help Centre

All users of our platform have access to a help centre on their home page or any page that they are on. They have access to FAQs to address any problems on the platform or how to fix any issues you may have. Along with that, there is a live agent option where they can reach a live agent to help them with any questions and problems they have.

When accessing the live agent option, they would have to wait in a virtual waiting room until an available agent can take them on to answer their questions and solve any problems they may be facing.

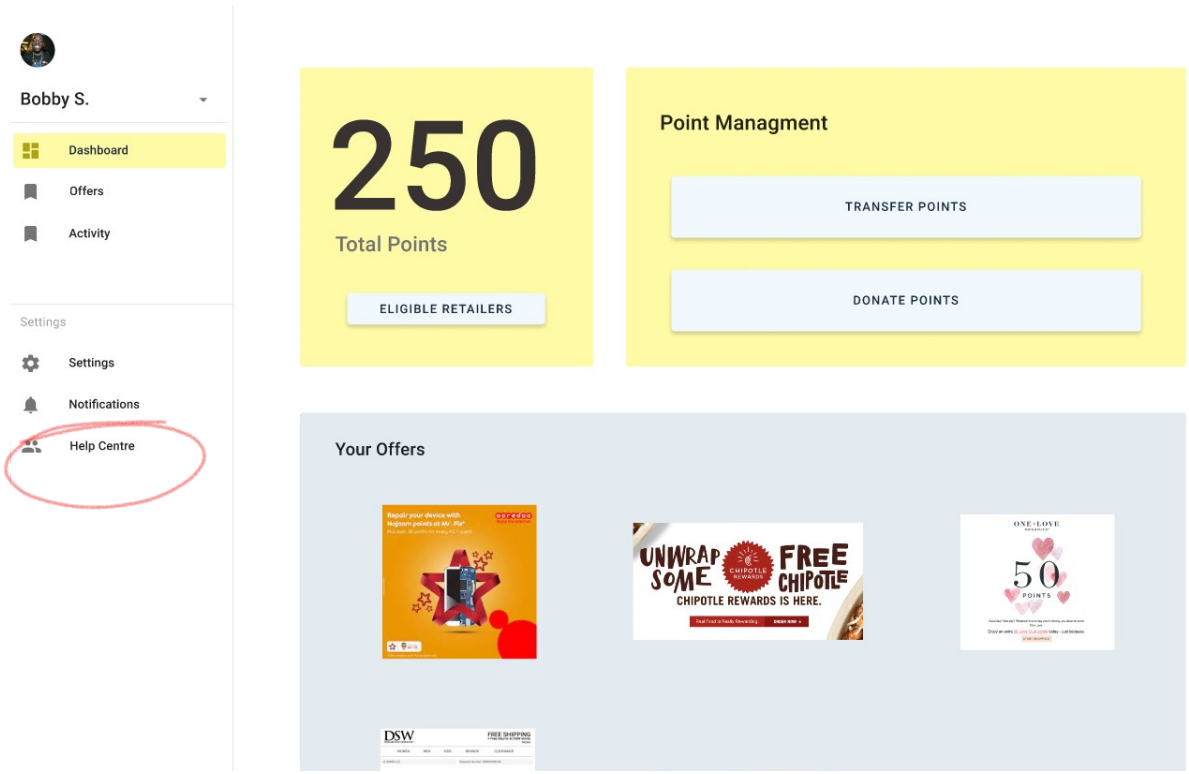

Figure 3. Where the help centre button is located on both the bank and retailer side of the application.

Bank Client's Side of Product

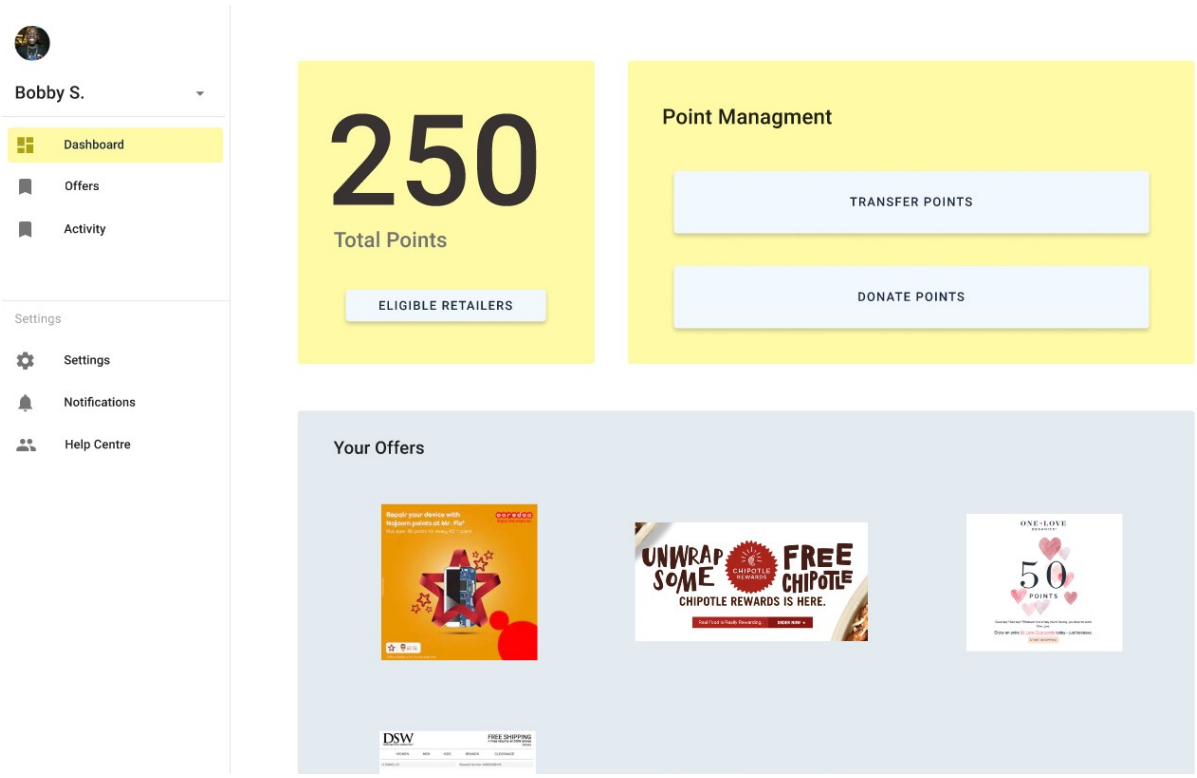

Figure 4. Bank Client's home page/dashboard.

View Total Points and Spend at Eligible Retailers

When regular bank clients log onto the platform, they are able to view the total points they have accumulated across all retailers. The total points that they can spend.(Refer to figure 4.) When you click on "Eligible Retailers", a user can view the dollars accumulated at each retailer in dollars, so they do not have to do the math of how much money they can save at each retailer.

This allows bank clients to know the amount of money they can save at each retailer all from one spot. They can click on each retailer they have accumulated points with so they can view any offers they have at each retailer as well.

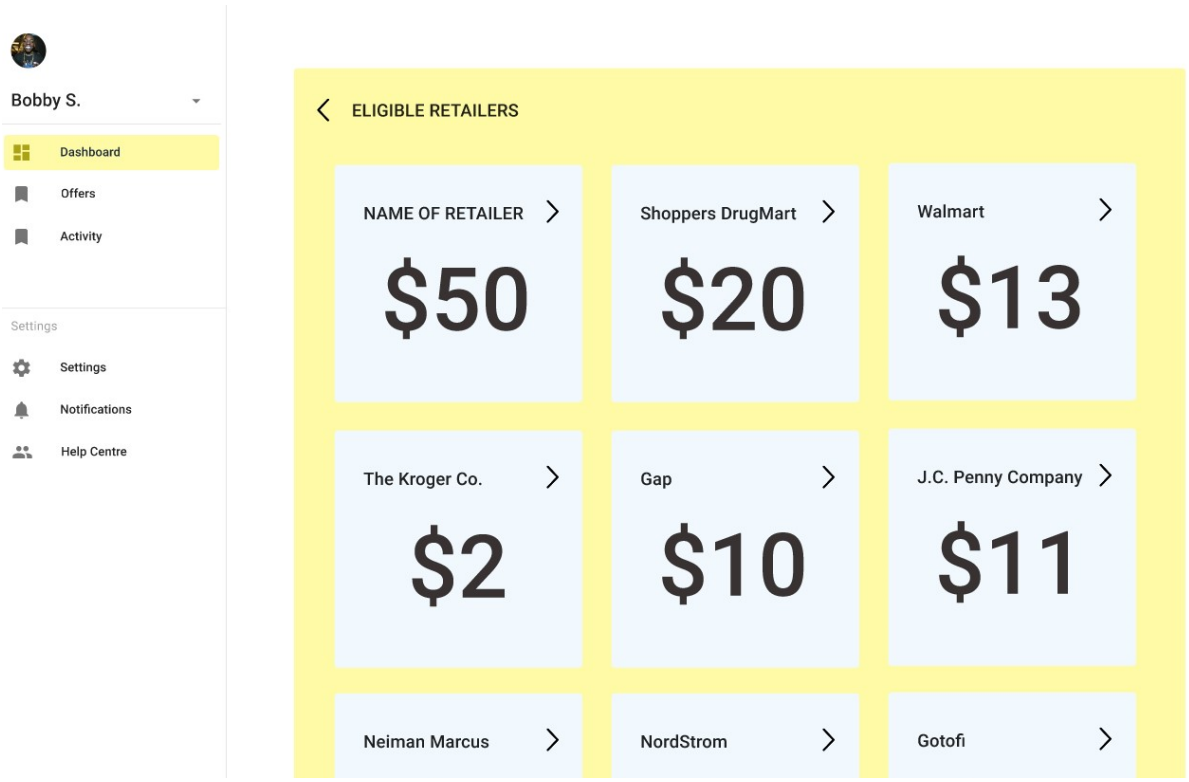

Figure 5. Eligible retailers page.

Point Management: Transfer/Donate Points

Bank clients are able to manage their points on this platform by clicking on "Transfer points" or "Donate points" on the application home page. (Refer to figure 4.)

Client's are able to transfer points from retailers they have points from to anyone who uses the same banking service as them. They can choose the amount and even add a message to their transfer.

Along with transferring points, clients can donate points to their favorite charities that are partnered with certain retailers.

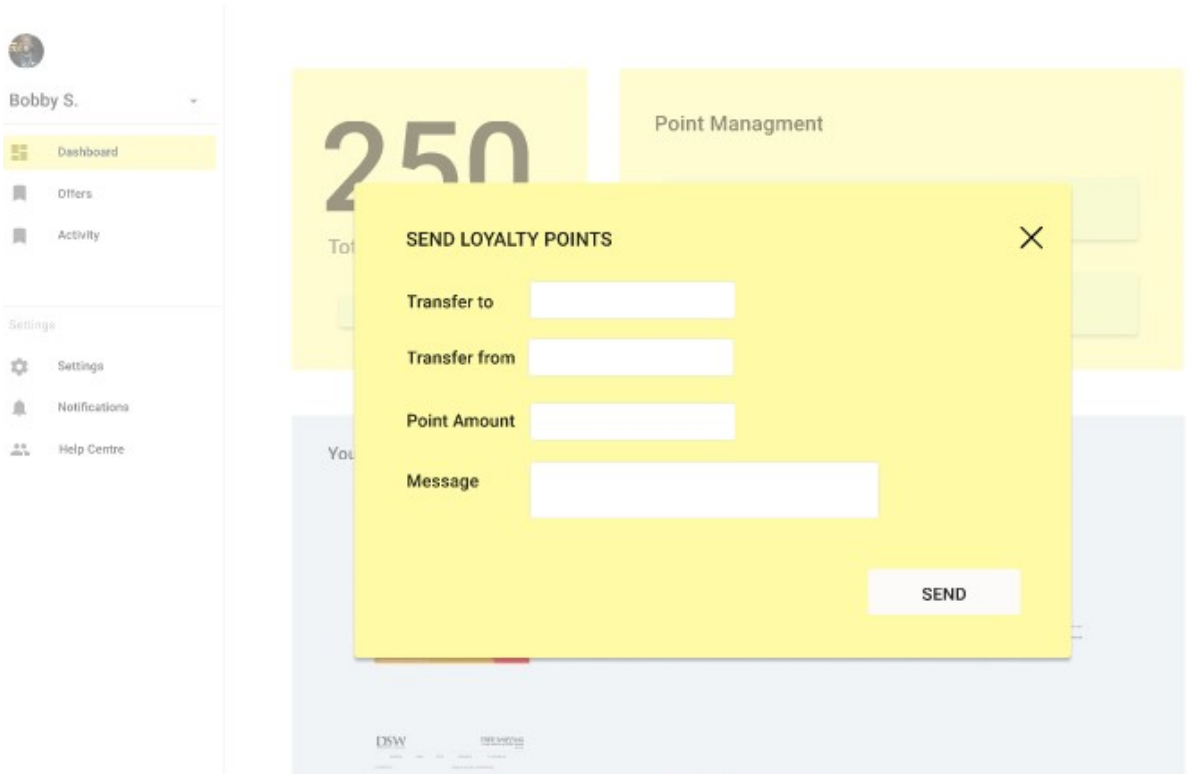

Figure 6. Transferring/Donating Points Page.

Personalized Offers on Dashboard

Users will be prompted with personalized overs on their home page. Showing products/offers that match their purchasing history with retailers and certain products. Providing users an experience that benefits them and their regular spending habits.

Also, they can edit their preferences to products/ retailers in settings if they are not satisfied with an ads that show on their personal home page.

To view all offers from all retailers registered with their bank, users can click the "Offers" button locate on the navigation bar to the left. This will lead a user to a page where they can search up retailers and see all the ads on this platform to browse any new deals/offers.

For personalized ads to show up on someone's home page, retailers will have to set their target audience when releasing an ad.(More on that under the *Retailer's Side of Product* section).

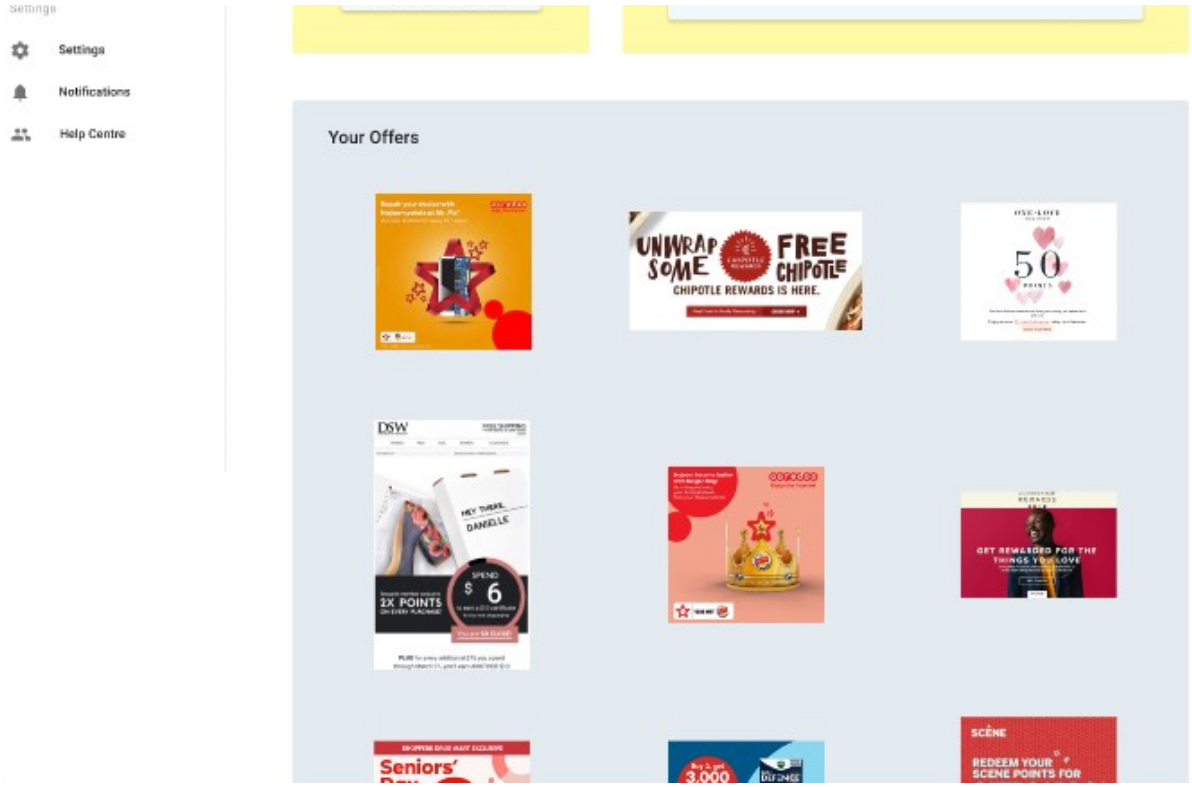

Figure 7. Personalized Offers on Home Page.

View Transaction/ Point Management History

Bank clients can view their transaction and transfer/donation of points history, all from one place. This makes it easy to refer to when checking on your financials. This is accessible when clicking on "Activity" button on the navigation bar to the left. (refer to figure 4.)

Retailer's Side of Product

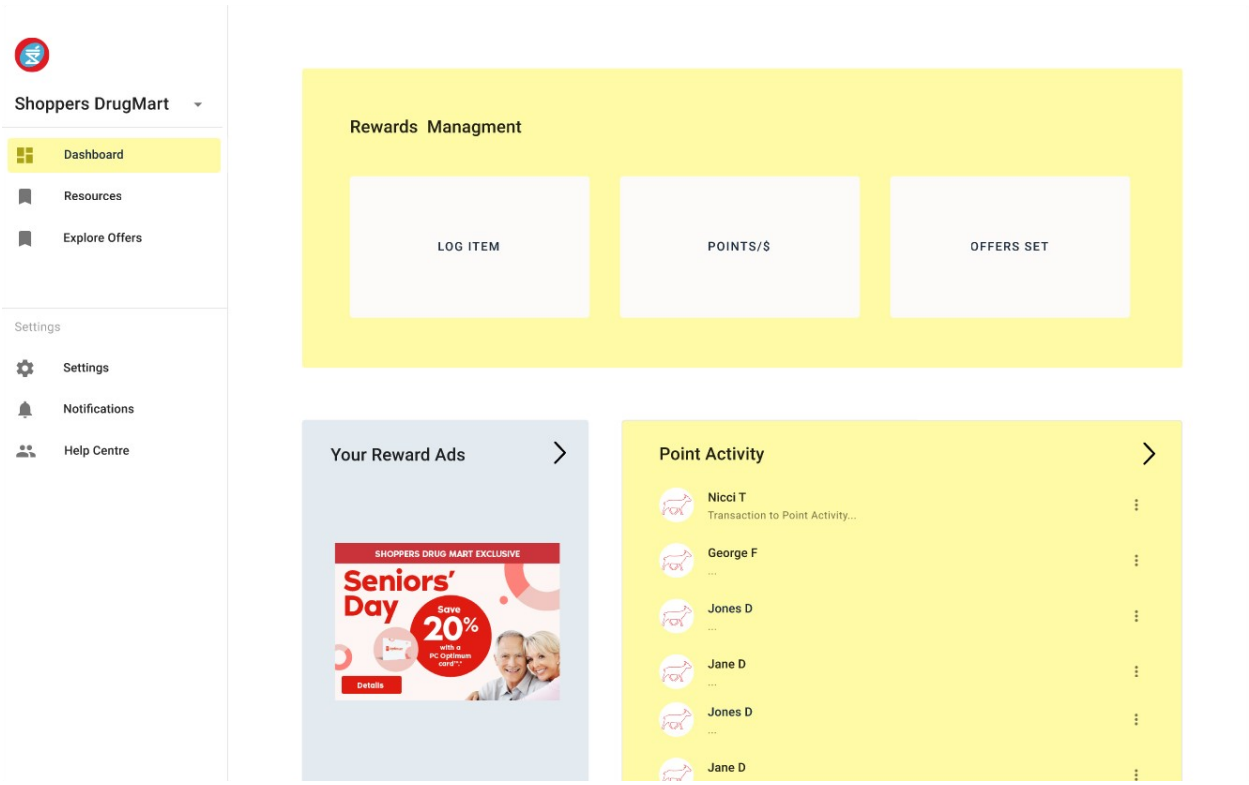

Figure 8. Retailer's home page/dashboard.

Manage Reward Program

Retailers are able to manage their loyalty rewards program where they can set their own points per dollar for their store, and log items into the system that are associated with points.

To set their points per dollar at all their stores they would have to access the "points/dollar" feature on their dashboard, where they can edit the points/$ value of their store.

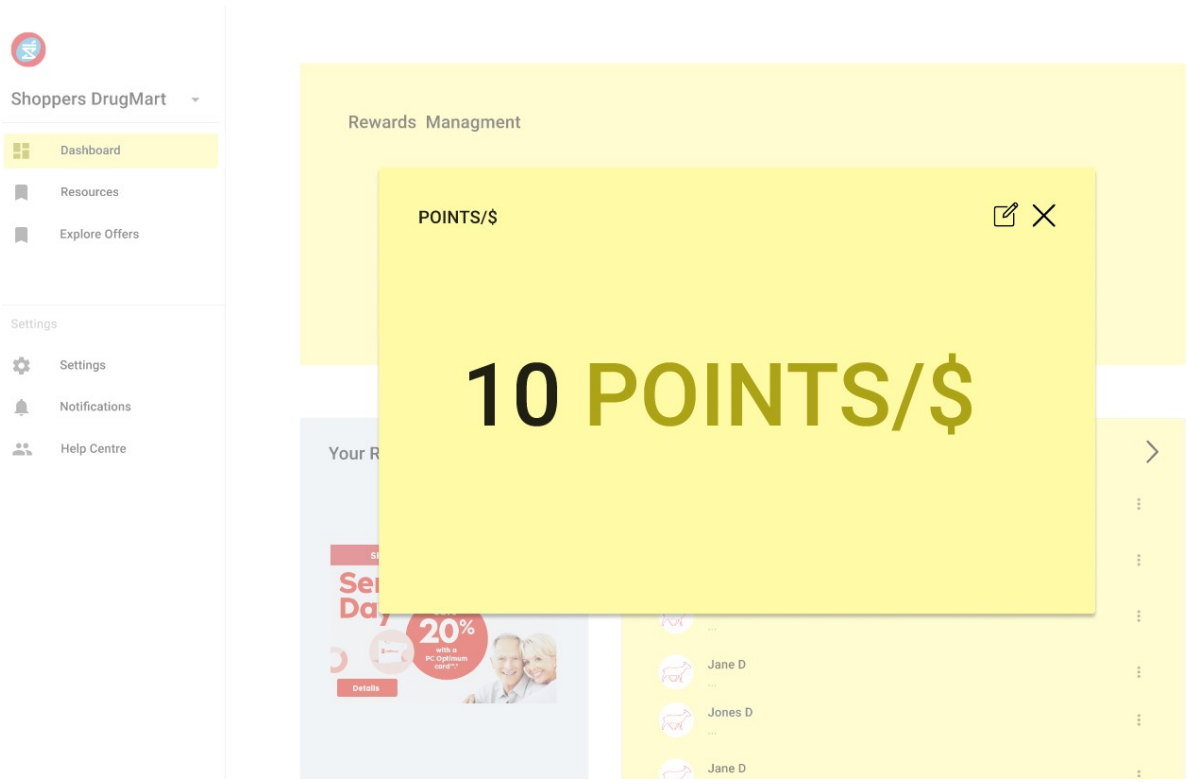

Figure 9. Point/$ interface.

Retailers can log items onto their system and set a point value to each item. They input the item name, number, category, price, and point/item. This makes it easy for the system to refer to items with a certain point value or with extra points due to offers.

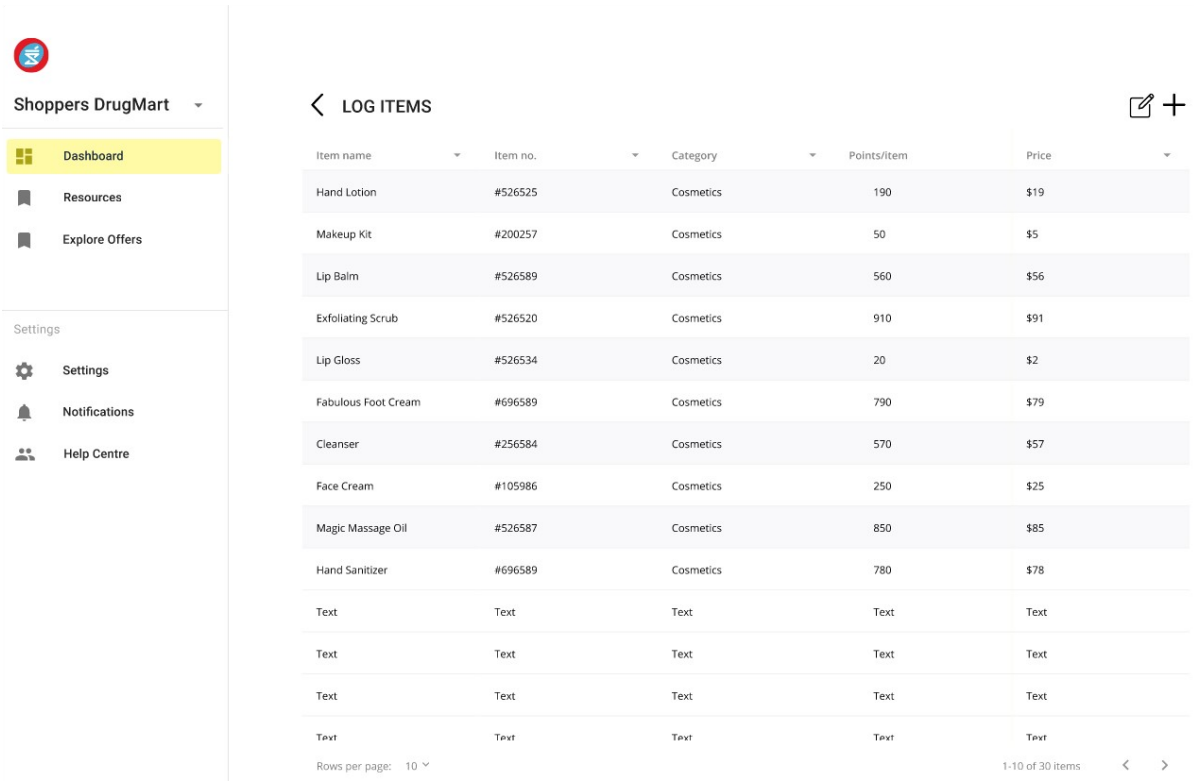

Figure 10. Logging items page. Keeping track of store inventory.

Set Reward Ads

Retailers can get more customers into their stores by setting ads with a target audience by clicking "Offers Set" or "Your Rewards Ads"(Refer to Figure 8 to see main dashboard interface) , and set ads just like google ads does. This can increase exposure to their business if done properly(help via resources provided- more on that later). Tags that describe your target customer's purchase history can help increase exposure to them. Our algorithm takes those tags and pushes that ad to customers' (bank client's) personalized offers page on their dashboard.

A retailer can ad their ad visuals, title, target audience, and a link to a page/ more information on offer if they would like.

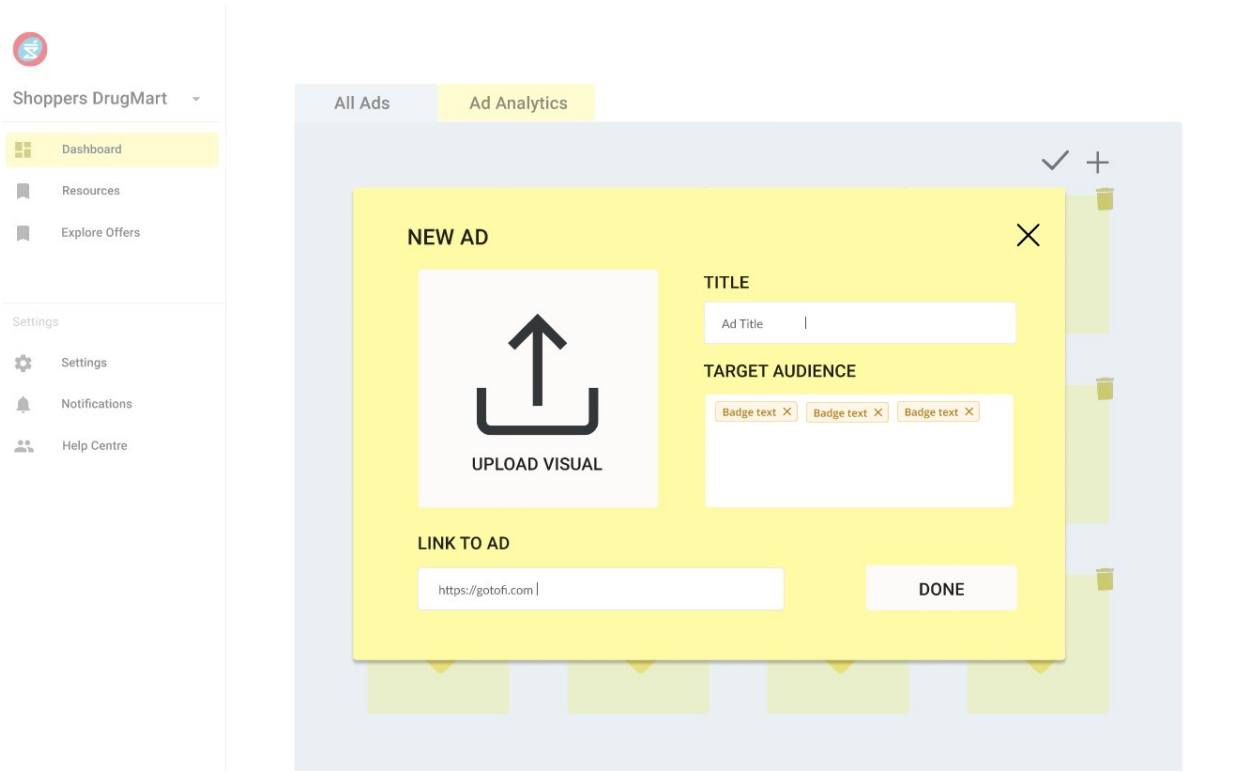

Figure 11. Creating a new ad interface to be pushed to target bank client.

A page with ad analytics can help a retailer analyze how their ads are doing. If a retailer does not understand how a successful ad campaign runs, they can check their resources to learn how to do this in a simple way.

They can see the number of clicks on their ad, any unique visitors, the number of people reached, and more. This information can help them improve their business by giving them feedback on their ads.

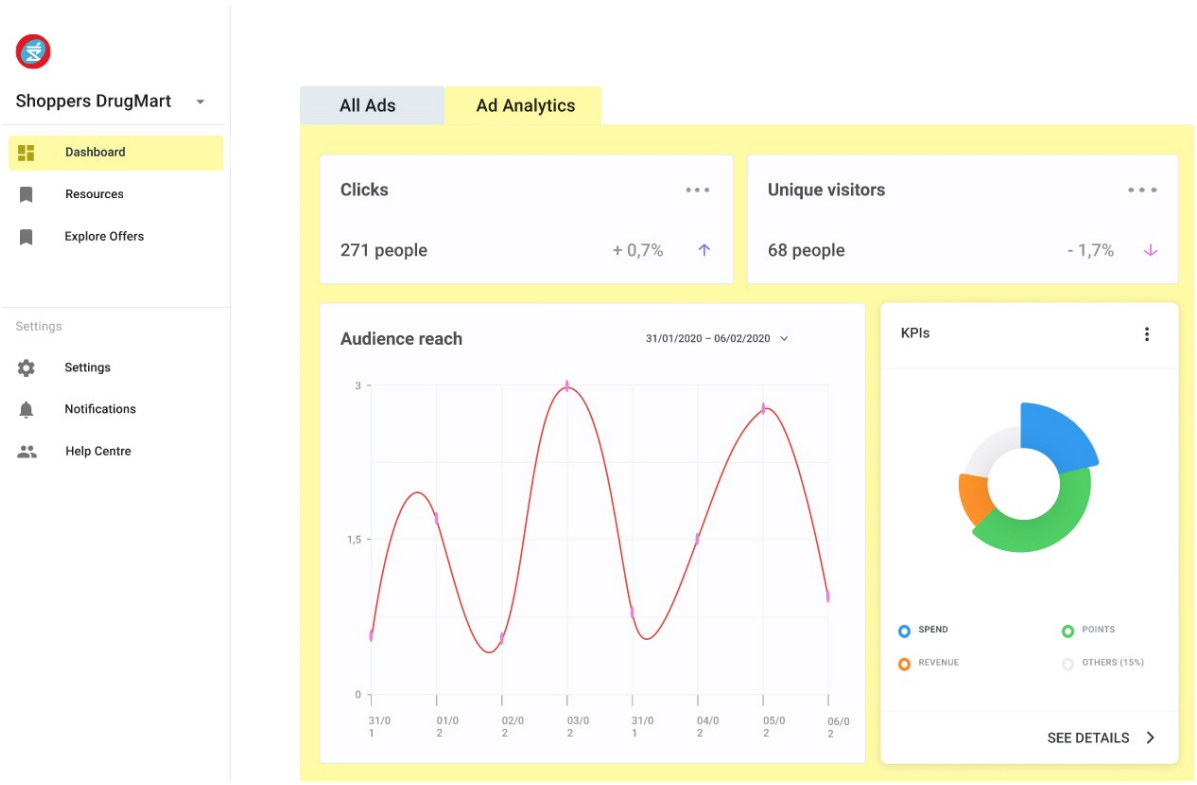

Figure 12. Ad analytics that retailers can analyze to improve their business.

View Store Transaction History

All customer transactions and points gained can be viewed by clicking on "Points Activity" on the main page. This is where you can view the names of customers, transactions, location of store shopped at, points gained/lost, and the amount spent. If for any reason there is a problem with transactions or point transfer it is good to have a history to refer back to, to be able to help out bank client's when they run into problems.

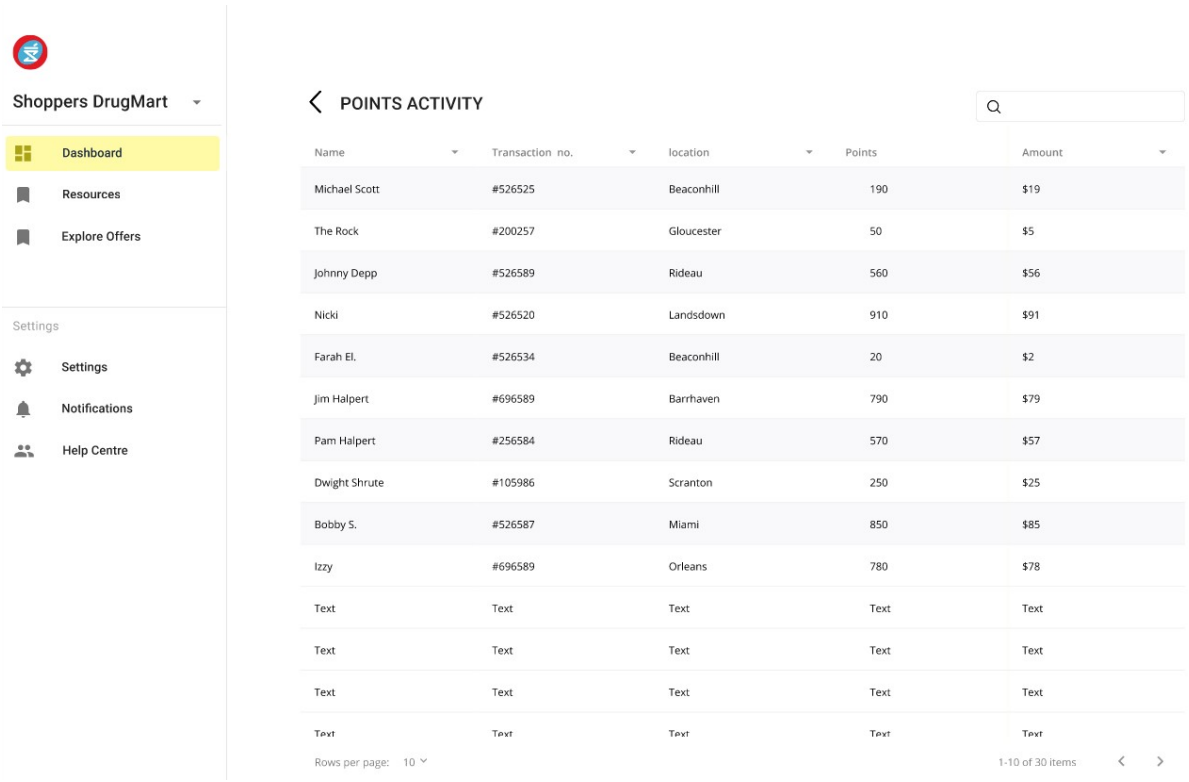

Figure 13. Point activity page to log all information.

Access to Resources

When clicking "Resources" on the navigation bar to the left of the screen, retailers are able to access resources that can help them run their business/ rewards program optimally using this platform. We understand that not every small business understands how to run a loyalty rewards program optimally, which is why this page exists, to bridge the gap in knowledge between big and small retailers. This will truly democratize the loyalty rewards industry as we try to level the playing field. The resources page should be able to help them understand how to optimize the usage of this platform, but if they can not find help on this page, they can access the help centre for personalized help.

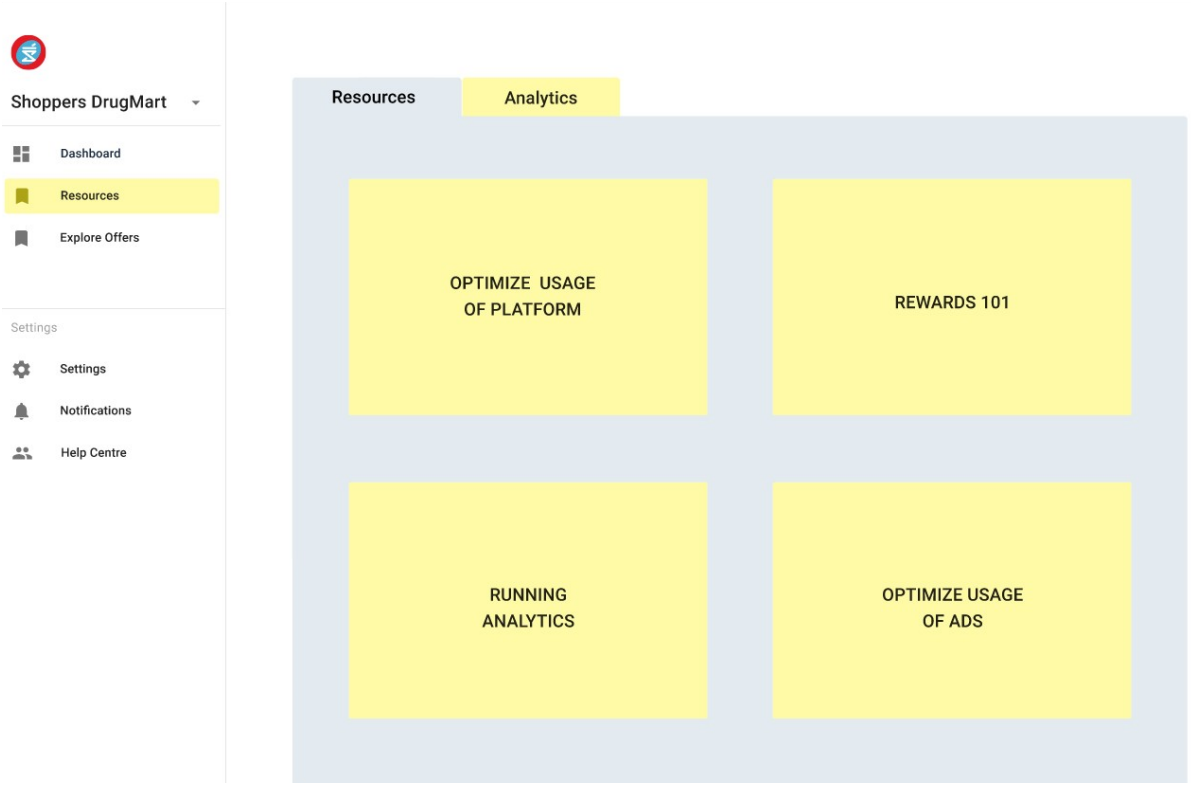

Figure 14. Resources page provide links to pages that walk through features and how to optimally use them. Personalized help can be accessed via Help Centre.

Troubleshooting and Support

Handling and troubleshooting errors for a platform connecting customers to retailers to banks, requires the utmost attention. In order to efficiently resolve issues, detection, categorization, and prioritization is very important since, the platform is dealing with a points economy, which can quickly be destroyed due to bugs, crashes, fraud, phishing and much more.

Error Messages or Behaviors

Error messages are integral parts of the platform, since, notifying the users of problems and potentially dysfunctional components is not only helpful for the users to correctly report these problems, but also for the team to solve them. Some examples, of potential error messages, include but are not limited to: unstable internet connection, failed transaction, unable to redeem, invalid login credentials. Of course, if the user input is

invalid, the system will allow the user to re-input the information, however, an error log will be made to make it easier to track fraudulent activity.

Special Considerations

The UI navigation and certain information load times can be exceptions, as updating the database and receiving information from the database may take some time, mainly from refreshing the information. Another limitation for troubleshooting is that the UX can only host locally on a system if the required development tools and API code is set up in the right configuration on the system.

Maintenance

The online database must be checked for any bugs in order to ensure that the points from the users' end are being added and deducted to their respective accounts upon spending and earning. Furthermore, the functionality for the merchant's page must also be checked in order to ensure that the items and the points the merchant associated with the se items are being properly displayed on the catalog page. Lastly, any errors that users may have encountered must be taken note of and thoroughly investigated to solve and prevent future errors from happening again.

Support

A help center will be present for all user, however, with a focus on the different features each type of user has. Therefore, regular consumers/customers will be able to access a FAQ section that will present an updated list of questions and their answers. Furthermore, a report section will also be available for the users to report the following: crashes, fraud, bugs/glitches, or others. If the user wishes to contact the help desk, they will need to navigate to the contact us page, where they will be able to find the phone numbers and email addresses they may use to voice their concerns. Out of all reports fraud will be the team's number one priority since it can be the most damaging to the reputation of the platform, the second priority will be to fix and bugs or glitches because, if a crucial functionality of the platform is compromised it can heavily damage the points economy as well as users. Crashes will be troubleshooted and fixed as quickly as possible, however, it will be not the top priority unless the amount of crash reports is abnormally high. This is because crashes may occur for many reasons such as incompatibility, insufficient system memory, unstable internet, outdated OS, etc... Due to

the variability of crashes, it is not the top priority in comparison to something less variable such as fraud or bugs and glitches. Retailers will have more contacting options and a different set of FAQs that pertain to managing and using the platform to benefit from different features such as the resource tab, points transactions, advertisements/monetization. etc...

Product Documentation

Needs Identification

Benchmarking

Technical and user benchmarking were used to measure other platforms' performance and try to fill out any gaps. We will provide you with two examples of benchmarking for PC Optimums and Nojoom Points Rewards.

Technical benchmarking:

- PC optimum:

- Earn points for certain items(ads on flyers, shelves, online portal for bonus items.)

- "Personalized" offers on PC Optimum App.

- redeem points for free groceries/merchandise

- 1,000 points= $1 CAD- redeemable in 10K increments.

- **Base Points Per Dollar Spent**: only at Shoppers and Gas Stations

- Most points come from special offers and bonus points

- Nojoom Points Rewards:

- Tier list split in Red, Silver, Gold and Al-Nokhba where the lowest would get you 15 points for every 4$ spent and the highest would get you 60 points for the same amount of money spent.

- Have different tiers of loyalty program members who have more valuable points.

- Tier list for customer is based on mobile usage prior to loyalty program enrollment. This is because Ooredoo(owner of Nojoom) is a telecommunications company

- Point redemption only limited to partners which include Fast-food restaurants, clothing stores and a few hotels as well

- Tier list split in Red, Silver, Gold and Al-Nokhba where the lowest would get you 15 points for every 4$ spent and the highest would get you 60 points for the same amount of money spent.

User benchmakring:

- PC optimum:

- People like: Anything that allows them to get bonus points(eg. point multiplier events, bonus points on certain purchases in the month,etc)

- People do not like:

1- Some people find it a hassle to have their PC card with them to make purchases, but most people don't mind as recently it can be added virtually(virtual card)

2- Sometimes people find the 10K increments of point redeeming to be tedious

- Nojoom Points Rewards:

- People like: The largest of its kind in Kuwait, and provides members with a unique opportunity to collect points and redeem them with the best offers and exclusive benefits.

- People do not like: Nojoom awards falling under Ooredoo has a very slow application that crashes very frequently. The customer service for loyalty points is slow and replies take a lot of time.

Problem

Until today, no platform democratizes loyalty programs and allows all retailers no matter their sizes to control or manage their own point system. Moreover, benefits and offers are limited in terms of redeeming for customers where they can not transfer, exchange or redeem points without limits. There is still no simple, accessible, and secure way that eliminates the need of using many platforms and cards to collect points and save money in regards to customer loyalty. Similarly, there is still no wide accessibility for smaller retailers where larger ones are dominating the market. To start our journey in finding a solution, we formulated a short, specific, and interesting problem statement and referred to it always throughout our progress. The problem statement is: ‘A need exists to create a highly secure, simple, and compatible platform that can allow users to

keep track of loyalty points, and their regular banking information for usage on various different retail stores, products, and services.’

Design Criteria

User needs were specified and translated into design criteria divided into functional requirements, non-functional requirements, and constraints.

Priority	Design Specification
Functional Requirements	
1	Authentication System for retailers and customers
2	Verify retailers/ Verify Bank Customers
3	Securely store points
4	Store history of redeemed points
5	Simple point gaining system (ability for customers to transfer, exchange and redeem points).
6	Convert purchases into points
7	Any retailer can sign on to the platform with ease
8	Retailers can manage/set their own point system
9	UI/UX- Clear display of information
10	Point calculator
Non-Functional Requirements	
1	Highly responsive
2	Has simple and intuitive navigation
3	Interactive and has a welcoming user interface
4	Available in various different languages
5	Scalabilty
Constarints	
1	Security and privacy
2	Reliability of program

Priority	Design Specification
3	User Interface/ User experience
4	Compatibility with all devices
5	Limit for spending points
6	Info-graphs on pages

Bill of Materials

BOM

Product	Cost	Links	Temporary Component
Figma	$15/month	https://www.figma.com/pricing/	final prototype 2
Notion	$5/month	https://www.notion.so/product	final
Wrike	Free	https://www.wrike.com/	temporary
MongoDB	0$ or $0.30/million reads	https://www.mongodb.com/pricing	final prototype 3
Node RED	0$	https://nodered.org/	final prototype 3
LucidCharts	120$/year	https://lucid.co/product/lucidchart	temporary
Firebase API/ libraries Reference	0$- free and open-source mobile UI framework	https://flutter.dev/docs	prototype 1 temporary
Flutter	0$- free and open-source mobile UI framework	https://flutter.dev/docs	prototype 1 temporary

How Our Final Solution Was Built

Our final solution is built into two high fidelity components: 1) the user interface 2) An app with a functioning backend.

The user interface is built using a tool called Figma. The main purpose of the Figma prototype is the show how the user interacts with the app, and how the final product would *look* like. Using Figma we were able to build an interface users can interact with , which simulates how it would be like to use the actual app.

This final prototype started out by building the bare-bone wireframes with all the desired features we would like to present to any user to test(bank client/retailer). This list included a user login page, verification page, bank client dashboard/offers/activity/ eligible retailers/ point management pages, and the retailers dashboard/resources/ offers/ ads/ rewards and points management pages

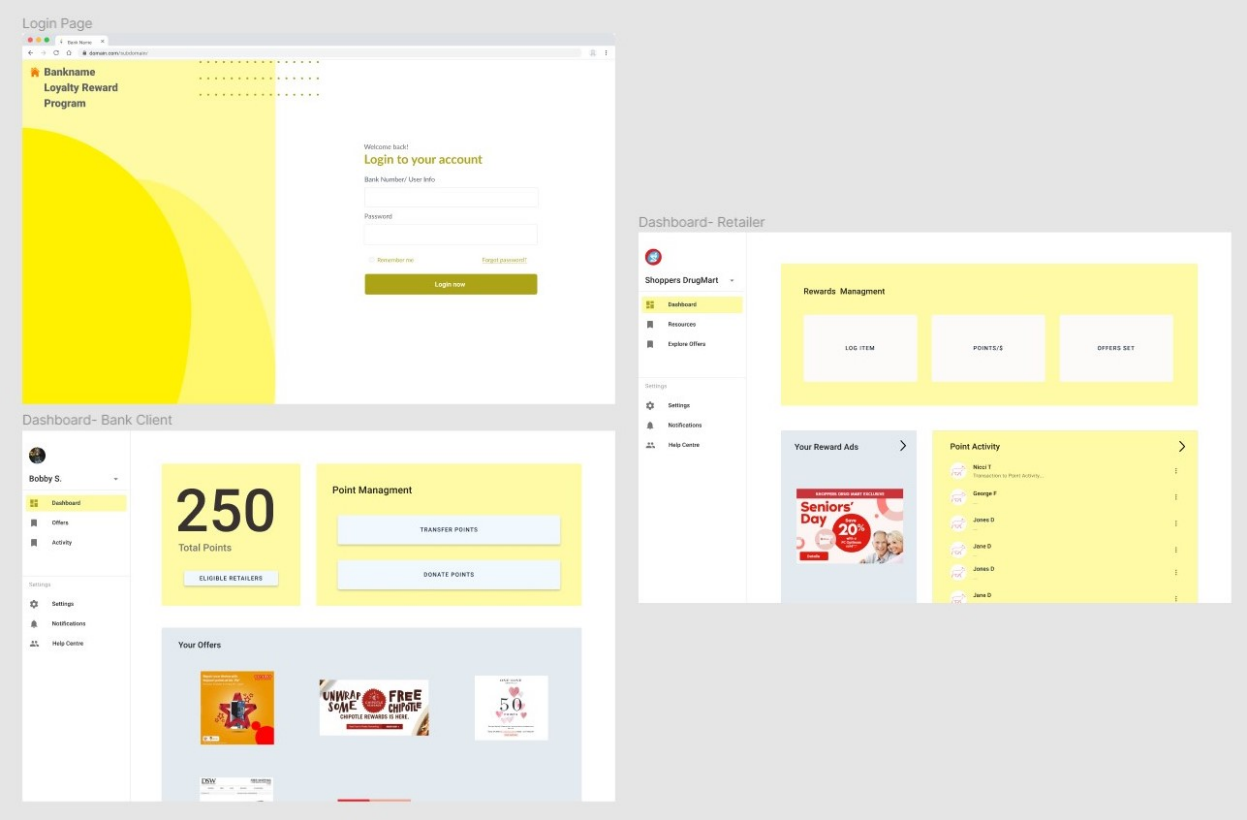

Figure 15. Initial wireframes with all desired features on Figma.

After having the bare design of the wireframes, we looked into the optimal way a user can access each feature and how their interaction with a feature should look like. A user should be able to access all features desired within 3 clicks, no more, to keep the program simple to use. This is when we added any animated "interactions with the

product like clicking on buttons, connecting pages together, and added editing and typing animations to simulate the management of points. This is all to simulate how interacting with the functional app should look like.

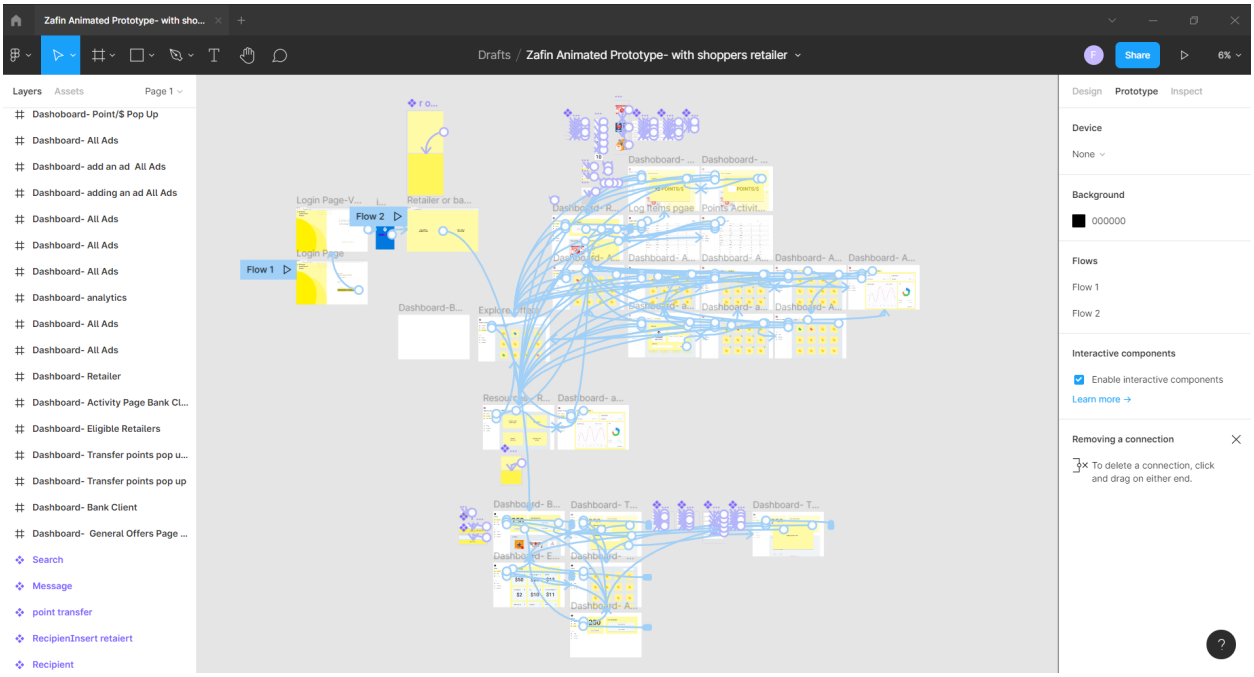

Figure 16. A look at all the wireframes and their interactions with each other on Figma.

This simulated the functionality of the app when interacting with the Figma prototype, but this had no backend to store the data of the app. To build a functioning app with a backend, we used NodeRed to build the frontend/ functionality and connected it to MongoDB to store the data.

NodeRed is a tool that uses flow-editors(refer to figure 20 and 22) to build programs in a visual manner to create JavaScript functions. This was used to create functioning APIs, such as one to ensure the right details have been entered when trying to log in, that convey information back and forth from front end to backend.

Before starting to create flows on NodeRed, first, we had to think about which data has to be stored, pulled, and referred to when users interact with the program. Then, think about how they interact with each other as part of the backend of the program.

To store the data to refer to, pull, or interact with in any way, we create tables/ a database on MongoDB that are connected to the NodeRED that the program can access while a user interacts with the frontend. (Refer to figure 21)

The purpose of the NodeRed/ MongoDB prototype is to show that we can build something functional that users can use.

Prototypes

Prototype 1

Prototype 1 consisted of a transaction number page, a tab layout, and a landing page for bank clients, retailers and customers. The transaction page and the tabs were constructed with the use of flutter in Android studio, where any required packages were installed.

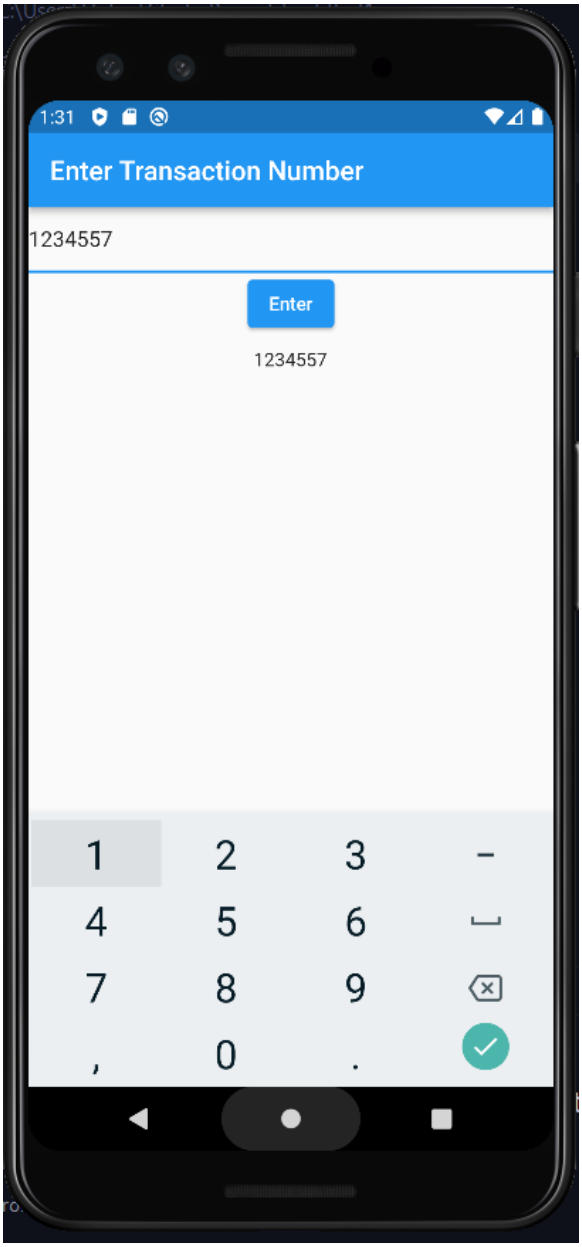

Figure 17: Transaction Number Interface

This user interface above allows entry of a particular transaction number that would be associated with a particular bank and particular item (e.g. Scotia bank and bread) and stores it. This value is made to be outputted to show that the system has accepted the integer transaction number.

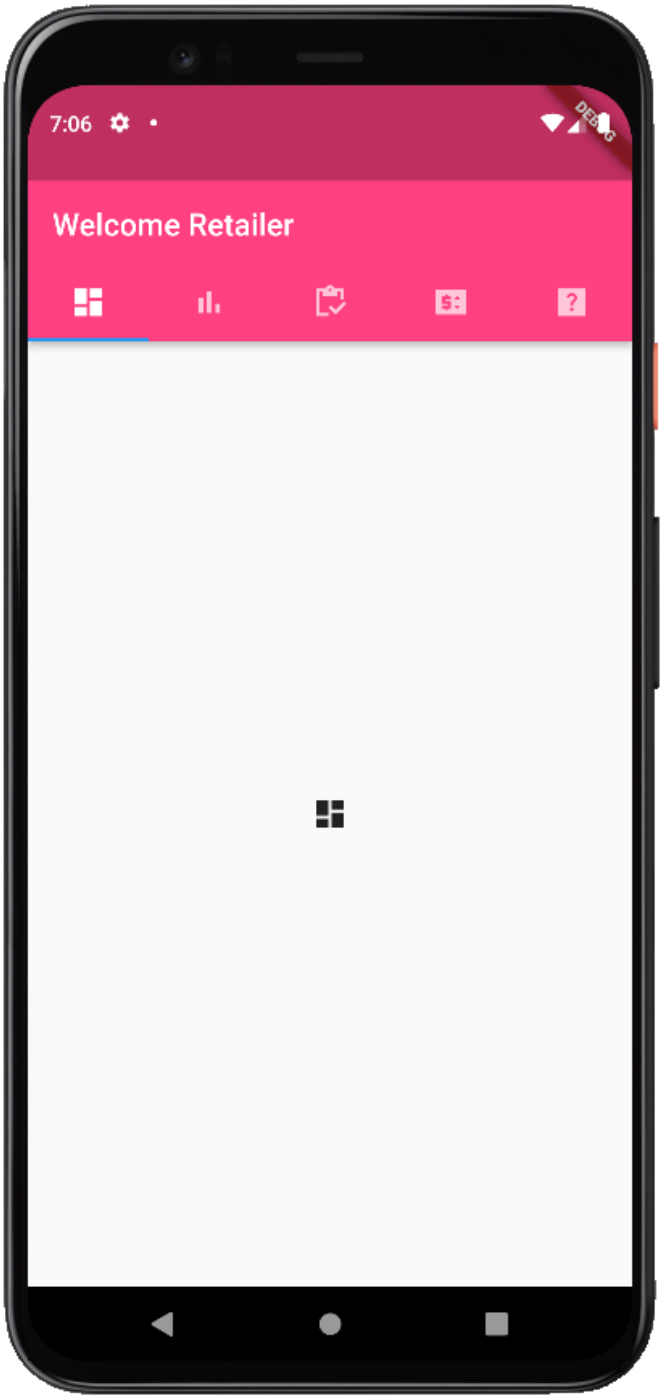

Figure 18: The page above is to be observed by retailers once they have logged into the system with their information. From this page, retailers will have control over multiple tabs which include a dashboard, a resource page, a points inventory, points transactions history, and a help center.

The dashboard will display a brief rundown of recent activities such as the net points related to the store, the store location, staff, manager, and any other information that

identifies the store owned by the retailer. The resource tab will display graphs that represent the data accumulated by the points history tab and also provide descriptive statistics such as an average loss/gain of points, general frequencies of how the points are spent and gained. The points inventory section allows the retailer to set points for possible trend items and also set point costs for rewards, etc... The points transaction history is the more in depth logging of every transaction made. Lastly, the help center will provide FAQs and other venues of seeking technical help.

A similar page is made for users which will have a dashboard containing tabs associated with viewing history of points, point redeeming, points collected for each retailer and based on that, offers available.

```
import 'package:flutter/material.dart';
import 'package:draw_graph/draw_graph.dart';
import 'package:draw_graph/models/feature.dart';
void main() {
  runApp(const TabBarDemo());
}

class TabBarDemo extends StatelessWidget {
  const TabBarDemo({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: DefaultTabController(
        length: 5,
        child: Scaffold(
          appBar: AppBar(
            backgroundColor: Color(0xFFFF4081),
            bottom: const TabBar(
              tabs: [
                Tab(icon: Icon(Icons.dashboard_sharp)),
                Tab(icon: Icon(Icons.bar_chart_sharp)),
                Tab(icon: Icon(Icons.inventory_sharp)),
                Tab(icon: Icon(Icons.price_change_sharp)),
                Tab(icon: Icon(Icons.help_center_sharp)),
              ],
            ),
            title: const Text('Welcome Retailer'),
          ),
          body: const TabBarView(
            children: [
              Icon(Icons.dashboard_sharp),
              Icon(Icons.bar_chart_sharp),
              Icon(Icons.inventory_sharp),
              Icon(Icons.price_change_sharp),
              Tab(icon: Icon(Icons.help_center_sharp)),
```

```
            ],
          ),
        ),
      ),
    );
  }
}

class GraphScreen extends StatefulWidget {
  @override
  _GraphScreenState createState() => _GraphScreenState();
}

class _GraphScreenState extends State<GraphScreen> {
  final List<Feature> features = [
    Feature(
      title: "Flutter",
      color: Colors.blue,
      data: [0.3, 0.6, 0.8, 0.9, 1, 1.2],
    ),
    Feature(
      title: "Kotlin",
      color: Colors.black,
      data: [1, 0.8, 0.6, 0.7, 0.3, 0.1],
    ),
    Feature(
      title: "Java",
      color: Colors.orange,
      data: [0.4, 0.2, 0.9, 0.5, 0.6, 0.4],
    ),
    Feature(
      title: "React Native",
      color: Colors.red,
      data: [0.5, 0.2, 0, 0.3, 1, 1.3],
    ),
    Feature(
      title: "Swift",
      color: Colors.green,
      data: [0.25, 0.6, 1, 0.5, 0.8, 1,4],
    ),
  ];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white54,
      appBar: AppBar(
        title: Text("Flutter Draw Graph Demo"),
        automaticallyImplyLeading: false,
      ),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        crossAxisAlignment: CrossAxisAlignment.center,
```

```
        children: <Widget>[
          Padding(
            padding: const EdgeInsets.symmetric(vertical: 64.0),
            child: Text(
              "Tasks Management",
              style: TextStyle(
                fontSize: 28,
                fontWeight: FontWeight.bold,
                letterSpacing: 2,
              ),
            ),
          ),
          LineGraph(
            features: features,
            size: Size(420, 450),
            labelX: ['Day 1', 'Day 2', 'Day 3', 'Day 4', 'Day 5', 'Day 6'],
            labelY: ['25%', '45%', '65%', '75%', '85%', '100%'],
            showDescription: true,
            graphColor: Colors.black87,
          ),
          SizedBox(
            height: 50,
          )
        ],
      ),
    );
  }
}
```

Figure 19: Above is the flutter code to achieve the retailer dashboard.

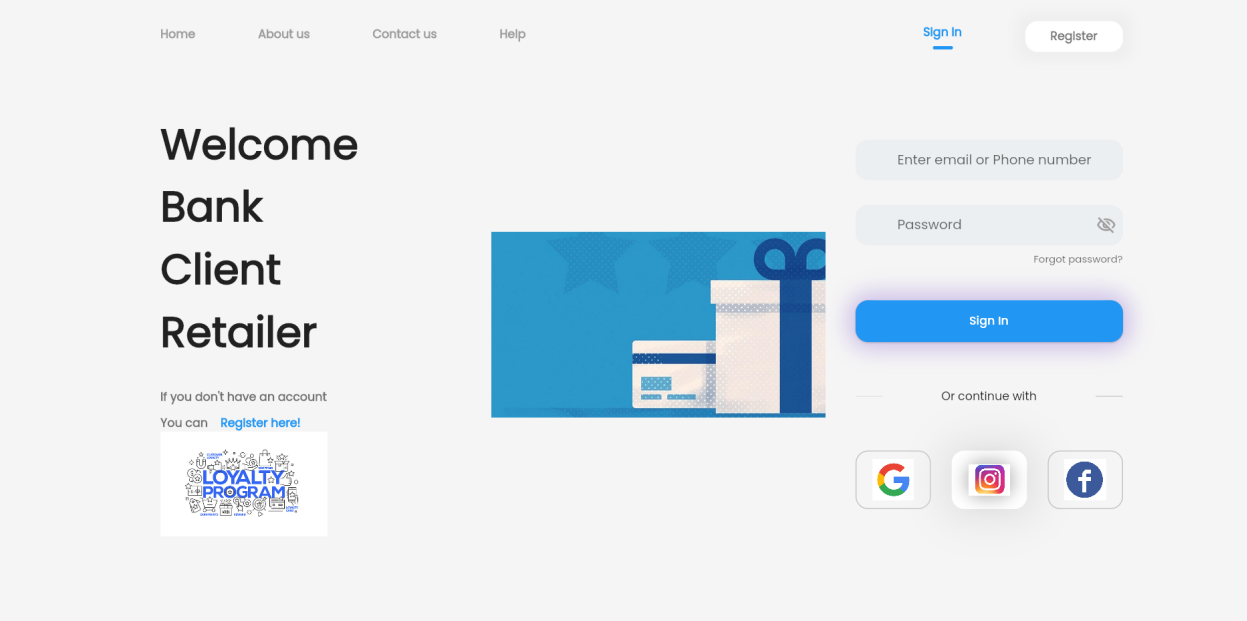

Figure 20. The page above is the landing page for bank clients, retailers, and bank employees. The user will be prompted to enter their email and a password which will allow login into their system and retrieve a personalized dashboard for the user.

Prototype 2

Prototype 2 marked a pivot in the team's choice of software from flutter to Node-RED and MongoDB as an online database. This prototype consisted of Figure 13-16 and a low-medium fidelity Node-RED program.

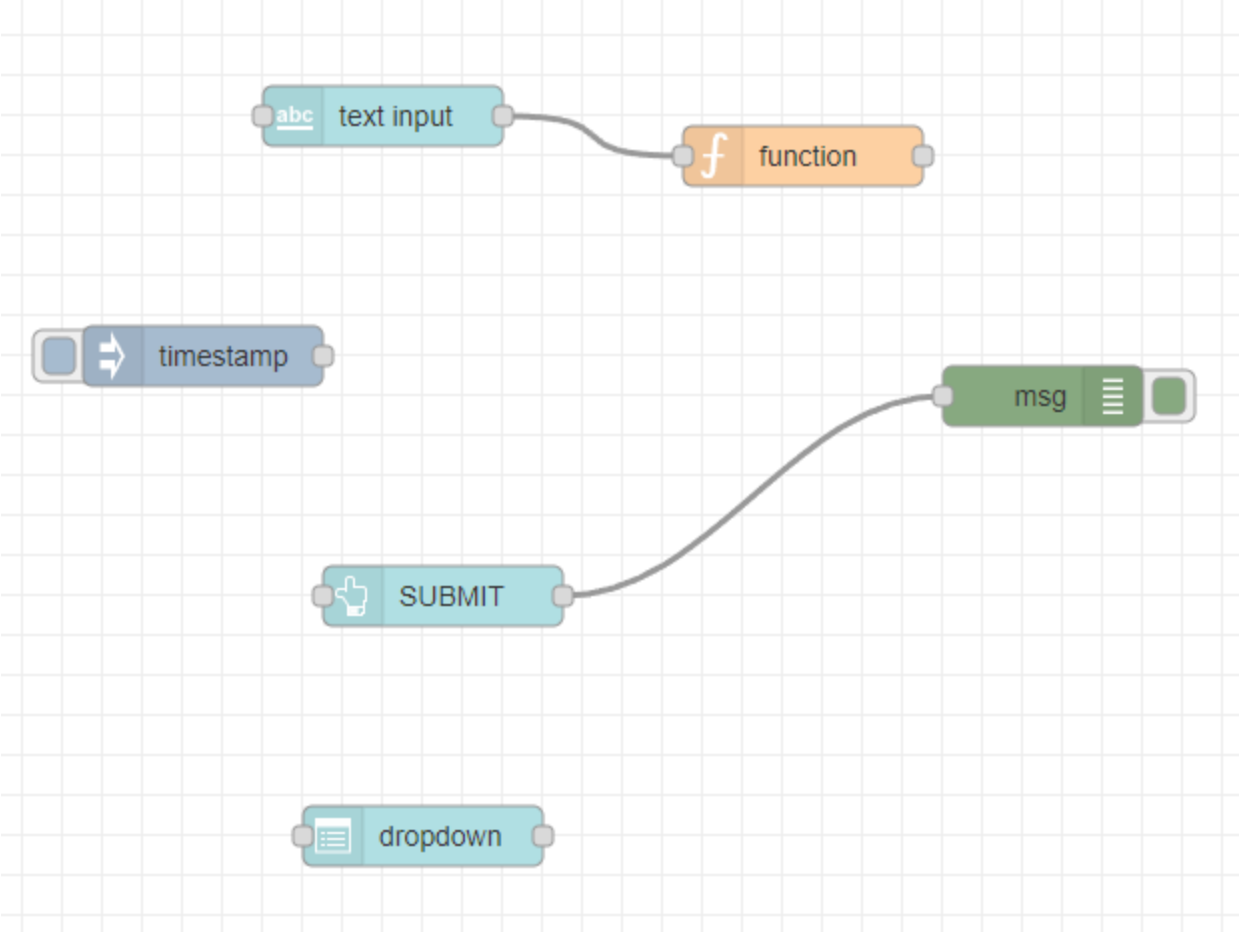

Figure 21: Node-RED Merchant item catalogue nodes

These nodes creates a drop-down menu that allows retailers to assign point to certain items. However, this prototype has not been connected to a MongoDB database yet, which has become the priority.

Prototype 3

Our prototype 3 was designed using node-red software. This allows users to use 'nodes' to carry out different functions to eventually build a UI dashboard. Our objective with this prototype was to create a dashboard where retailers can allocate points to each category of product at their store, and bank clients to redeem and store points. In order to do this, we created a database with test data and different APIs on Visual Studio that linked the node-red software to the database.

MongoDB databases:

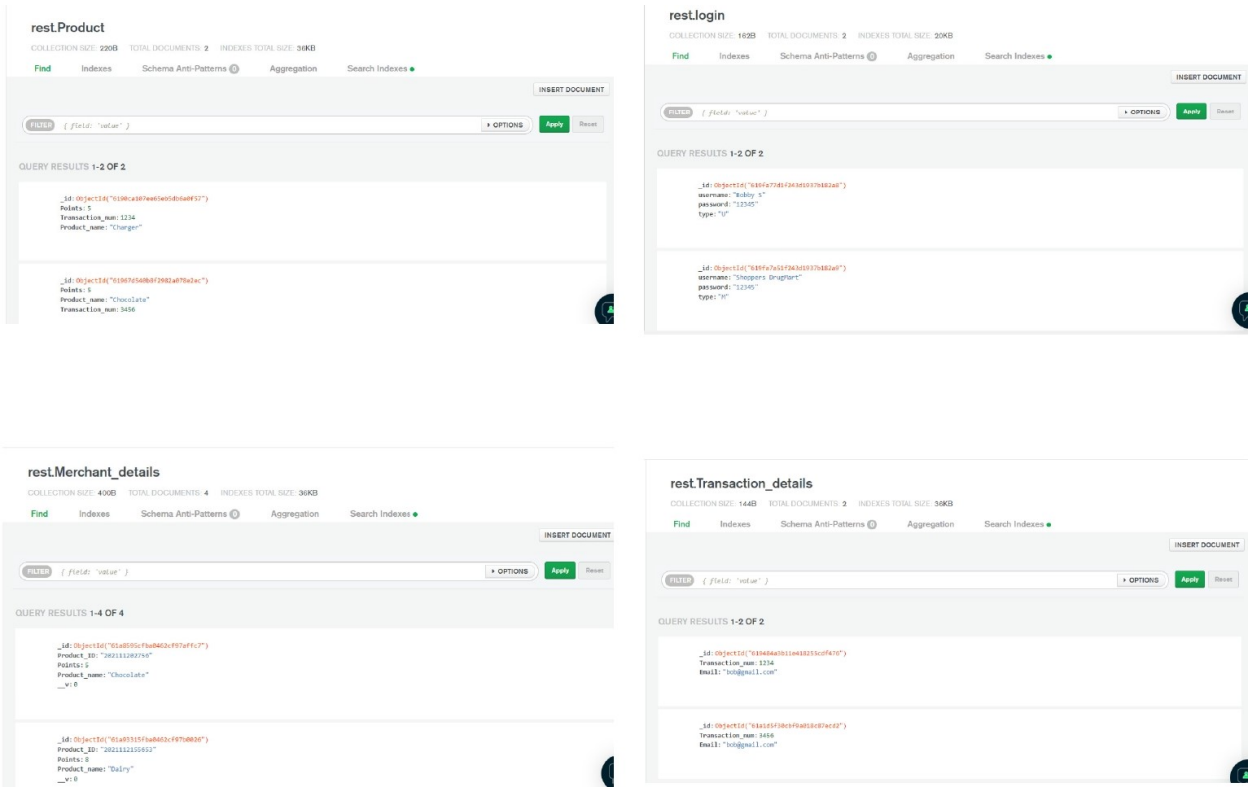

Figure. 22.a MongoDB databases

As seen above, we created four different databases to store different information. The product database stores information about the product name, transaction number, and points associated. The login page contains information about the username and password, and whether the user is a retailer or bank client. The Merchant details database contains a unique product ID, product name, and points associated. Finally, the transaction details database contains transaction numbers and email ID details. It's quite clear that sections in one database and found in another and hence, the databases are all connected to each other. This cannot directly be accessed until an API is created.

Login page UI nodes

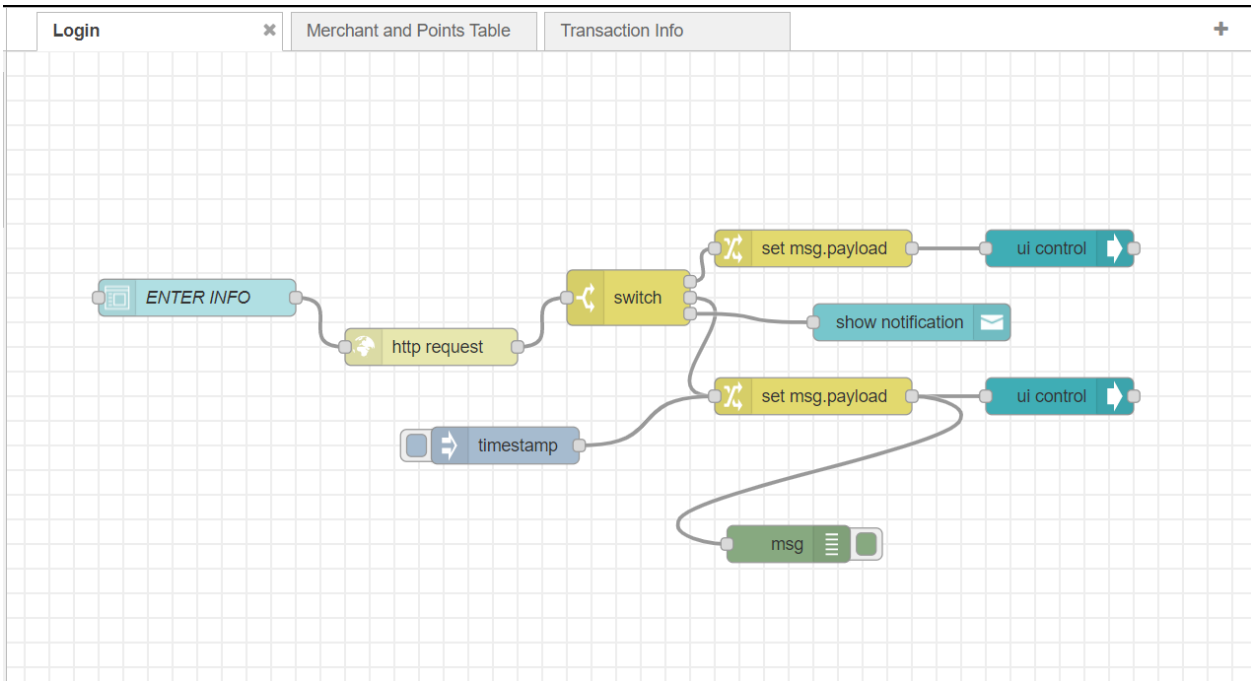

Figure 22.b. Login UI nodes

The Node-red software allows us to create a flow of processes that we want the login page to accomplish. The light blue node allows user input, which then leads to the calling of an API that retrieves login information from the database. If this input information is equal to the one provided on the database, the tabs will switch accordingly depending on if the user is a retailer or bank client.

Login information API

```
const Login = require('../models/login.js');

exports.loginDetails = (req, res) => {
    Login.findOne({
        "username": req.body.username,
        "password": req.body.password
    }).then(login => {
        if (!login) {
            return res.status(404).send({
                status: 1
            });
             }
   return res.status(200).send({login
});
    });
}
```

Figure 23. Login information API

Above is the API that compares the input from the user with the login information on the database. A value of U is returned if the user is a bank client and M is returned if the user is a retailer. The corresponding switch node decides what tab to switch to next. If neither condition is met, a value of 1 is returned.

Login page UI dashboard

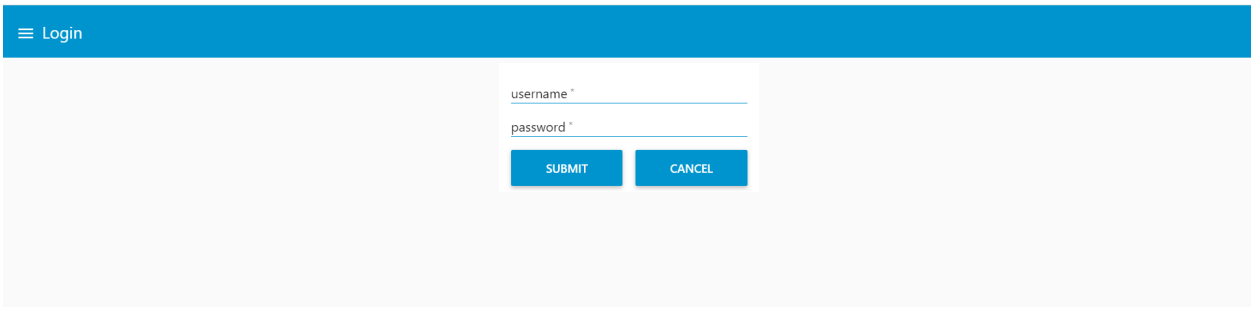

Figure 24. Login page UI dashboard

And hence, the login page that will be displayed to the user is displayed above. If the login database is examined, a username of 'Shopper DrugMart' and password of '12345' corresponds to retailer's login information.

Transaction info UI nodes

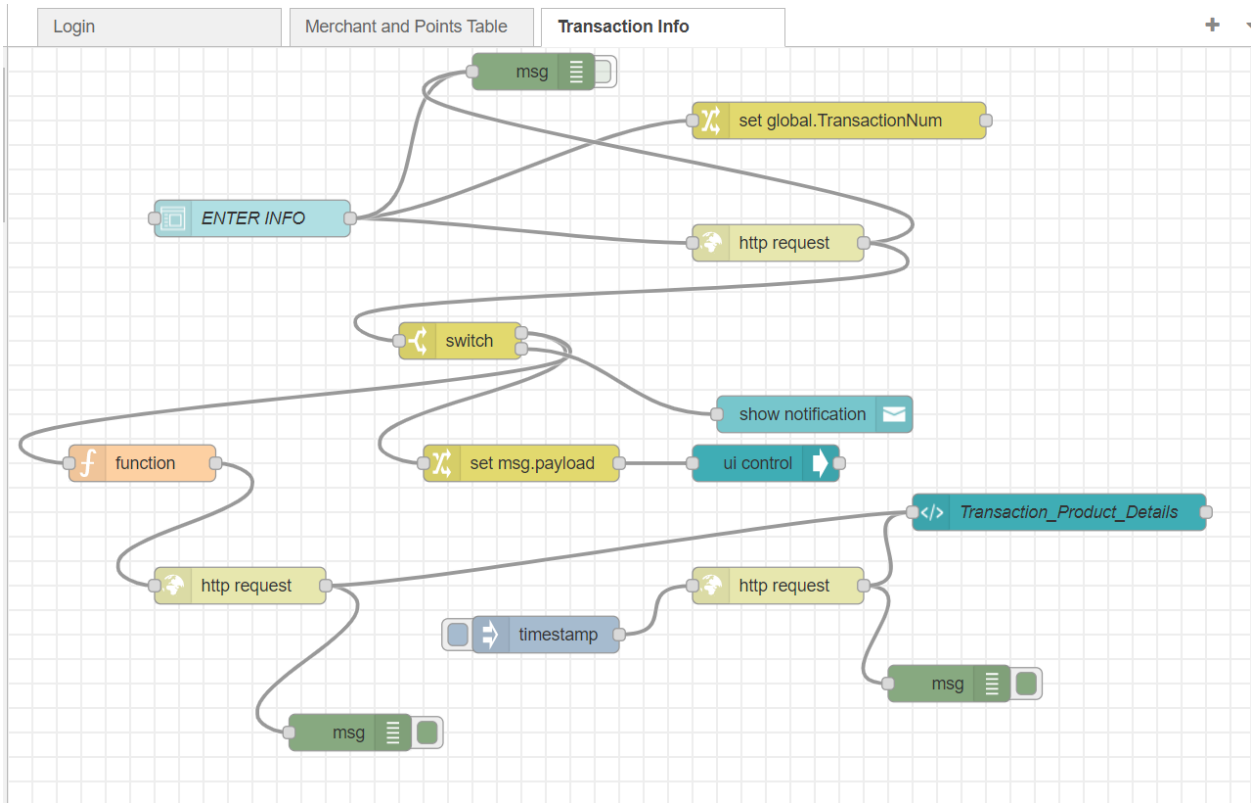

Figure 25. Transaction info UI nodes

Although daunting at first, the placement of the nodes is very systematic. This page is the next page that will be shown after the login page for bank clients. As seen before, the user is prompted to enter information. in this case, information about the transaction number and email for verification. Again this leads to the calling of a different API that displays the product and points added to the bank client's account.

Transaction validation and product-points retrieval API

```
exports.getTransaction = (req, res) => {
    Transaction_details.findOne({
        "Transaction_num": req.body.TransactionNum,
        "Email": req.body.Email
    }).then(Transaction => {
        //console.log(JSON.stringify(engine));
        if (!Transaction) {
            return res.status(404).send({
                status: 0
            });
             }
   return res.status(200).send({
                status: 1
});
    });
}

exports.getProductDetailsUsingTransaction = (req, res) => {
  console.log(req.body.TransactionNum)
    Product.find({
        "Transaction_num": req.body.TransactionNum
    }).then(Transaction => {
        console.log(JSON.stringify(Transaction));
        if (!Transaction) {
            return res.status(404).send({
                status: 0
            });
             }
   return res.status(200).send({Transaction});
    });
}
```

Figure 26. Transaction validation and product-points retrieved API

The API above is responsible for firstly ensuring the transaction information is correct and corresponds to the one mentioned on the database, and secondly retrieves and displays the product name and points earned.

Transaction page for bank client dashboard UI and error handling

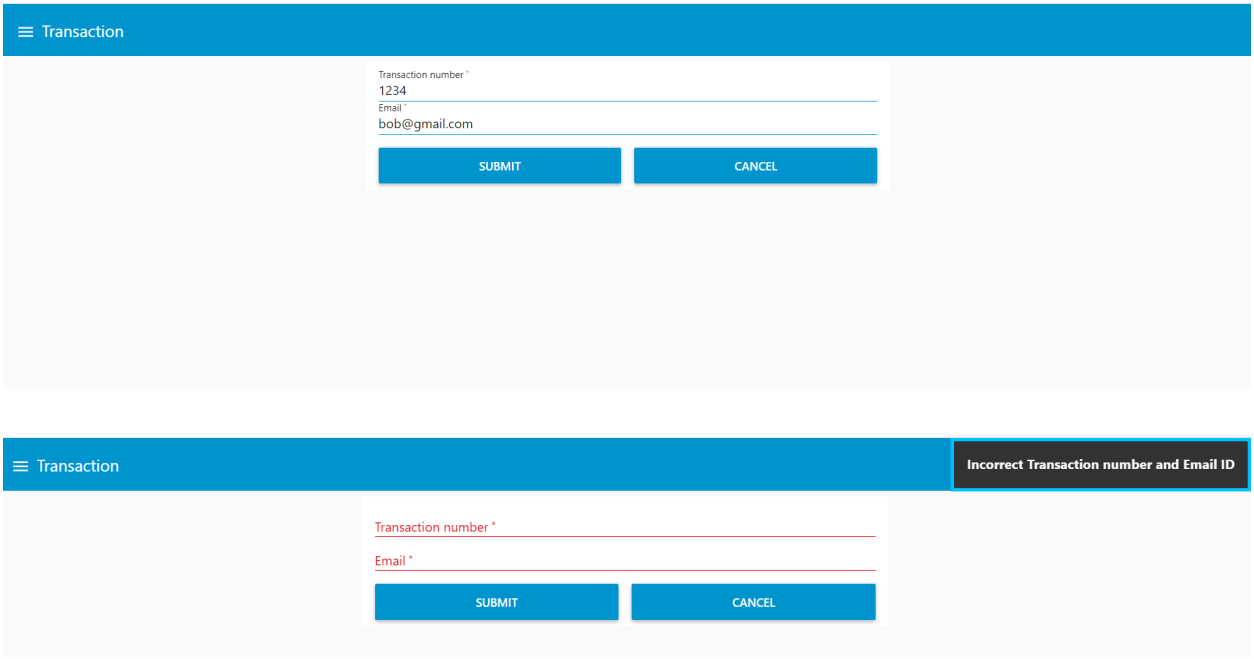

Figure 27. Transaction page for bank client dashboard UI and error handling

the correct details entered will lead to a display of the points table. As shown above, the correct transaction number and email are 1234 and bob@gmail.com respectively. The Transaction number type-in box only accepts integer input and the Email box allows string input. If any of these conditions are not met, an error message will be visible to the user as shown above(Incorrect Transaction number and Email ID). To improve, we will be focusing on the UI aspect of the application since functional requirements have been met.

Bank client points table UI dashboard

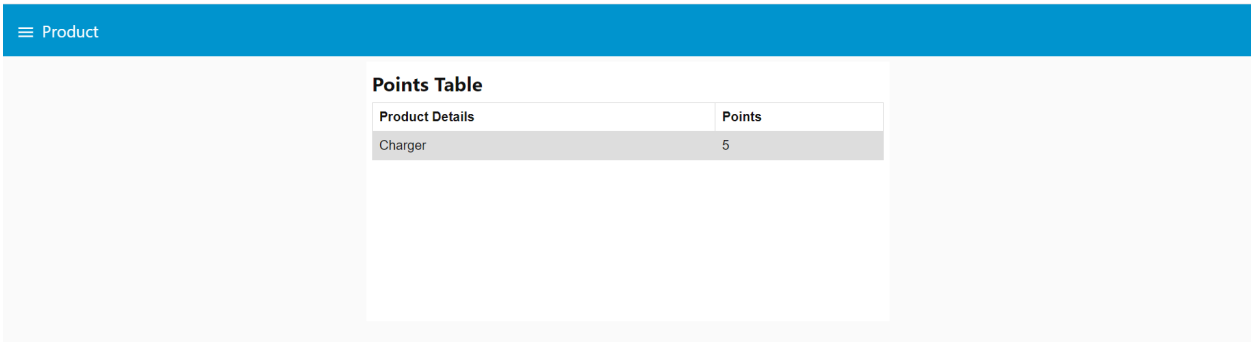

Figure 28. Bank client points table UI dashboard

The products table contains the details of the product corresponding to the transaction number. This is retrieved from the MongoDB database where information for a particular item is contained in a table. The transaction number 1234 corresponds to a charger and has associated with it 5 points.

Retailers page UI nodes

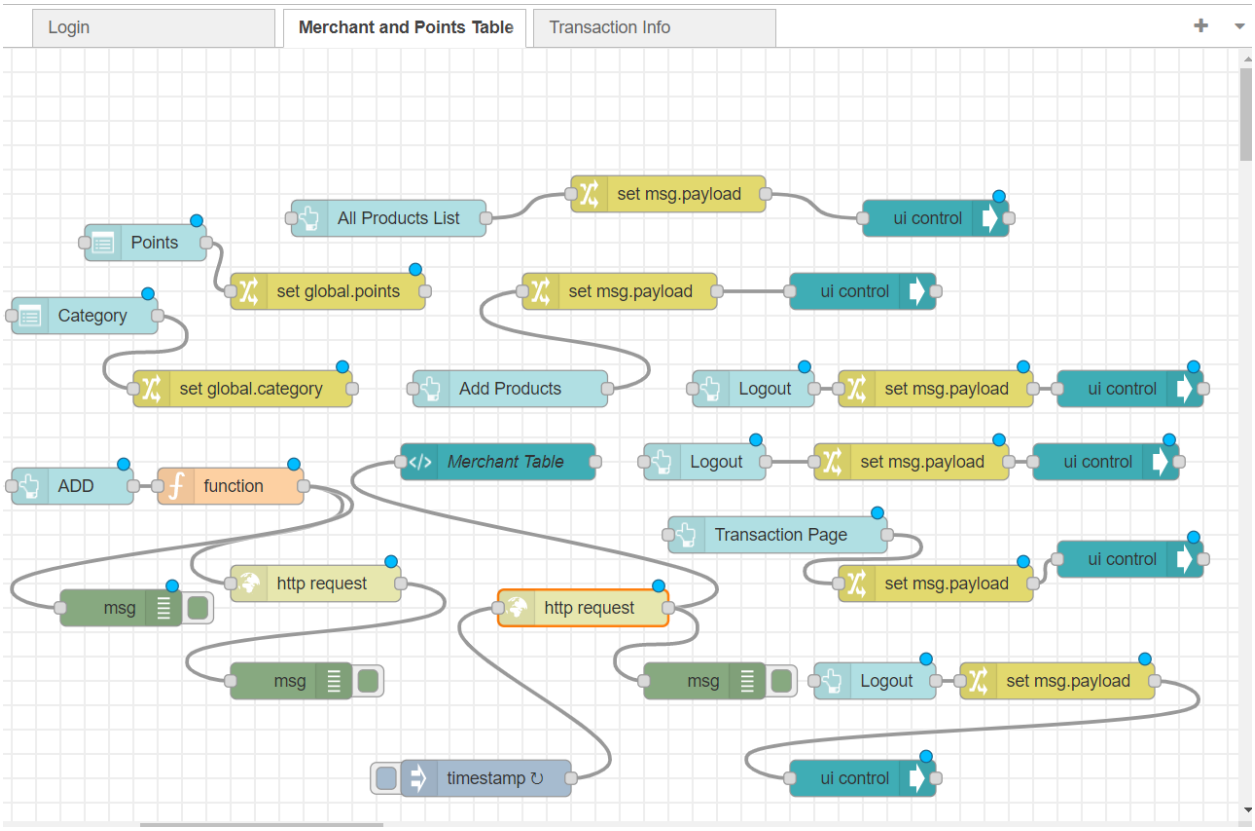

Figure 29. Retailers page UI nodes

In this specific flow, there are various features. Firstly, to improve the User experience, we used a light blue node to display a logout button to exit the program and return to the login page. As seen above, multiple logout nodes are visible as they are displayed on different pages. On the retailer's page, the http node highlighted calls the contents of the merchant points table in the database and displays the information. On the same page, the 'Add Products' button node when pressed, takes the user to another page where points can be allocated. On this merchant page, two dropdown nodes names Category and Points are visible. Each category has been filled with different dropdown options that the retailer has the freedom to choose. Once selected, the 'ADD' node is pressed.

This leads to a function which calls the API to add selection to the 'Merchant_details' database. Finally, an 'All Products List' button has been added to easily navigate back to the initial retailer page.

Retailers Points table

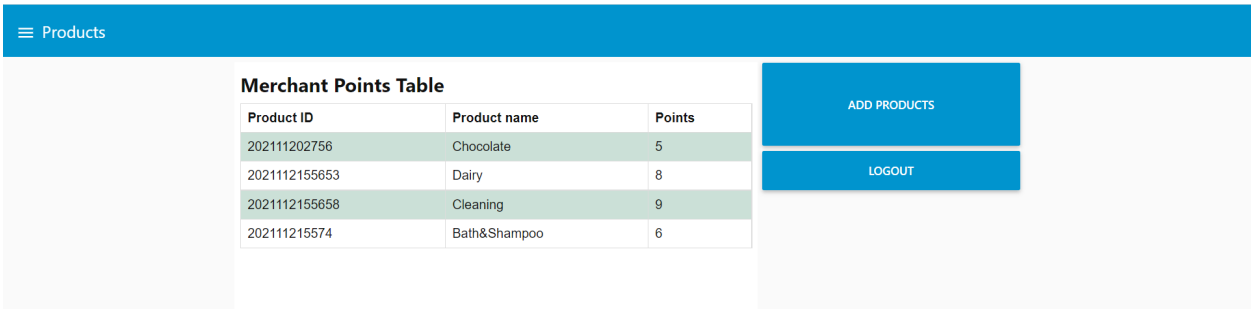

Figure 30. Retailers Points table

After retailers have successfully entered login details, the table above will be visible which is edited once the 'ADD PRODUCTS' button is clicked. In the current page, retailers has the option to either log out or continue setting points to each category of product. A unique Product ID is associated to ensure updated point entries are different.

Product-points add dashboard UI

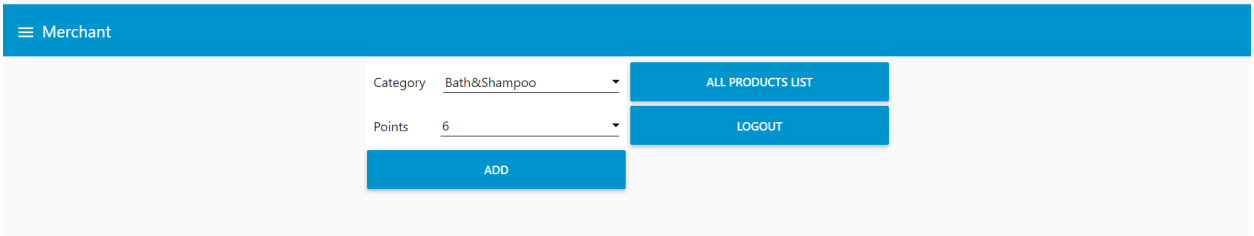

Figure 31. Product-points add dashboard UI

Retailers will be able to select exactly what category and points are rewarded from a dropdown box. Once the 'ADD' button is clicked, the information has been added to the database and can then be displayed on the Merchant Points table shown previously.

Merchant details-add and retrieve API

```
exports.create = (req, res) => {

    // Create a Note
    const merchant_details = new Merchant_details({
        Product_ID: req.body.Product_ID,
        Points: req.body.Points,
        Product_name: req.body.Product_name
    });

    // Save Note in the database
    merchant_details.save()
    .then(data => {
        res.send(data);
    }).catch(err => {
        res.status(500).send({
            message: err.message || "Some error occurred while creating the Merchant details."
        });
    });
};

// Retrieve and return all notes from the database.
exports.findAll = (req, res) => {
    Merchant_details.find()
    .then(Merchant_details=> {
        res.send(Merchant_details);
    }).catch(err => {
        res.status(500).send({
            message: err.message || "Some error occurred while retrieving Merchant details."
        });
    });
};
```

Figure 32. Merchant details-add and retrieve API

This JavaScript code allows products, points, and a unique product ID to be entered into the system database. Contents added can then be retrieved from the database and displayed in the form of a table.

Testing and Validation

The procedure of creating the primary prototype with the use of Node-Red and MongoDB was re-created on a new system, which allowed for a collaborative effort on the completion of the prototype as the second system was used for testing purposes.

The testing was performed on both the original version of the prototype and on a test version that is located on different computers. As there is a delay in delivering the code and implementing it in the test version of the prototype, testing does not happen simultaneously but rather with 1-2 days of delay. The testing, therefore, verifies that the prototype can run on multiple devices simultaneously. Testing and troubleshooting

transactions, point redemption, and management which are back-end oriented is being handled by team members that are doing the UX testing and development. Meanwhile, UI refinements and any UI software changes are being handled by the front-end-oriented team. While some UI aspects that are integral for testing the back-end code and database connection will be checked and refined by the back-end team. Validation of the main functionalities and the ability of the online database to handle multiple systems simultaneously is successful.

Conclusion and Recommendations for Future Work

In conclusion, this user and product manual helps in providing a deeper overview of our product and interprets all the steps that were taken to build it. Our loyalty-program-related platform will guarantee limitless applications for bank customers where they will be able to transfer, exchange, and redeem points using only our app and their bank cards. Moreover, retailers of all sizes will be able to sign onto our app where each can set and manage their own point system. Following this user manual, all readers will be able to use the platform with ease, and go through the troubleshooting and support section when experiencing any errors. While building the platform, the design thinking process was followed and applied in which benchmarking different products, setting design criteria, and more importantly, understanding the problem and formulating a good problem statement took place. Three detailed prototypes were presented to show how we came up with the final functional platform using NodeRed and MongoDB, and a functional user interface using Figma. All functioning apps were tested and validated following a test plan, and they are well illustrated using proper documentation.

Significantly, our app interface only shows how the full-functioning final product will look like so it is recommended that readers and users always check for updates and improvements. We will be working on finalizing the platform where users can greatly enjoy the new approach of loyalty programs.

References

N/A

Appendices

Please do check our maker-repo where we have documented our journey:

https://makerepo.com/HatimShakir/1012.a8-points-do-not-lie