National College of Ireland

BSc (Honours) in Computing

Software Development

2020/2021

George King
18106188
X18106188@student.ncirl.ie

SwiftSwipe

Technical Report

Contents

Executive Summary

The purpose of this report is to cover all aspects of my final year project in software development. The main points of this report cover the requirements of the system, the architectural design of the system, code implementations and GUI designs. I have yet to reach any conclusions at the midpoint or have any recommendations.

1.0 Introduction

1.1. Background

I undertook this project because last summer I had the chance to conceptualize the idea during a hackathon but did not get the chance to develop beyond the idea. I see this project as the opportunity to do that. I also think the application has real world potential and a strong likelihood to be a disruptive technology to the traditional way of shopping.

1.2. Aims

When looking at the aims of my project there are many that come to mind. The most important aim of my project is to make the customer experience seamless. The success or failure of my application relies on providing value to said customer. If I am unable to do this, it is unlikely that stores will be hypothetically interested in buying my application. There is the possibility that my aim of reducing store employees via the functionality of my application would also be attractive to potential buyers. When thinking about who my direct customer is for my application, which would be stores, it is possible to say that if I am successful in reducing employees in stores the application could still be considered successful. Overall, the aim of my project is to innovate and disrupt the current status quo of retail stores.

1.3. Technology

The first piece of technology that I used is Node.js. This technology is used for executing JavaScript code outside of a web browser. I utilized this technology to take and refund payments within my application. This helped me achieve my goal of taking payments anywhere in the store and reducing employees required in the store.

The next piece of technology that I used for the project is Java. Java is an object orientated programming language that I have been working with for five years. My strong abilities with this language give me a key advantage in being able to construct the application I wanted. I used the Java language alongside android studio to develop the front and backend of my application. This helped achieve my goal by facilitating a user interface for employees to interact with.

For my database I used firebase. I have picked this database for several reasons with the first being familiarity with the software. Having used the database in the past with various projects I am comfortable knowing it will be able to provide all the functionality I need. The next reason is the scalability of the database. Firebase allows a user to scale as much as they want and only charges for end user interaction. The final reason I choose firebase was its compatibility with my technology stack which was a key consideration when looking for a database. I also used firebase CLI software to evaluate the databases performance at the end of the project.

The final piece of technology that I used is Junit. Junit is a testing language that works well with the Java language. With any application it is important to ensure that functions within the application work as intended. In order for me to ensure this I decided to use the Junit language. Alongside Junit I used espresso, a step through testing language to ensure my layouts were working. I was also able to use this test language to test some of the more complex functionality in my system.

1.4. Structure

For the system section of this report, I have looked at several topics. Included in this is all the requirements for the system, the GUI of the system, the design of the system and finally the architecture of the system.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram

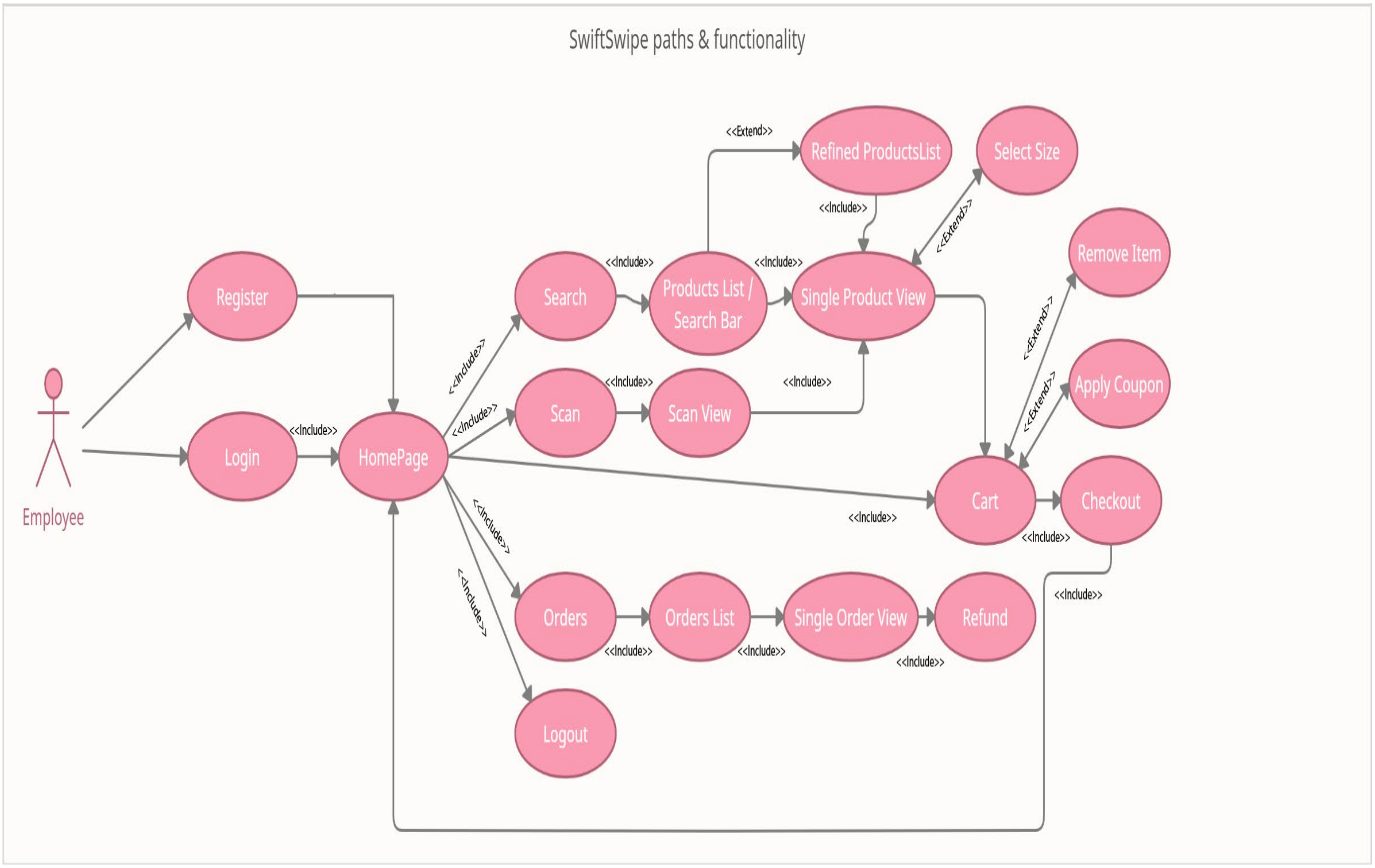


Figure 1 Overall system use case diagram

2.1.1.2 Requirement 1: Employee searching the stockroom

2.1.1.2.1 Description & Priority

The requirement is about the employee using the associated database with the store to find items stored in it. Due to this being a fundamental aspect of my application I have given it a priority of high.

2.1.1.2.2 Use Case

Scope

The scope of this use case is to show an employee using the database to find an item after having logged in.

Description

This use case describes the process in which an employee logs into the application and either searches for an item manually in the database or scans a barcode to find the item in the database.

Use Case Diagram

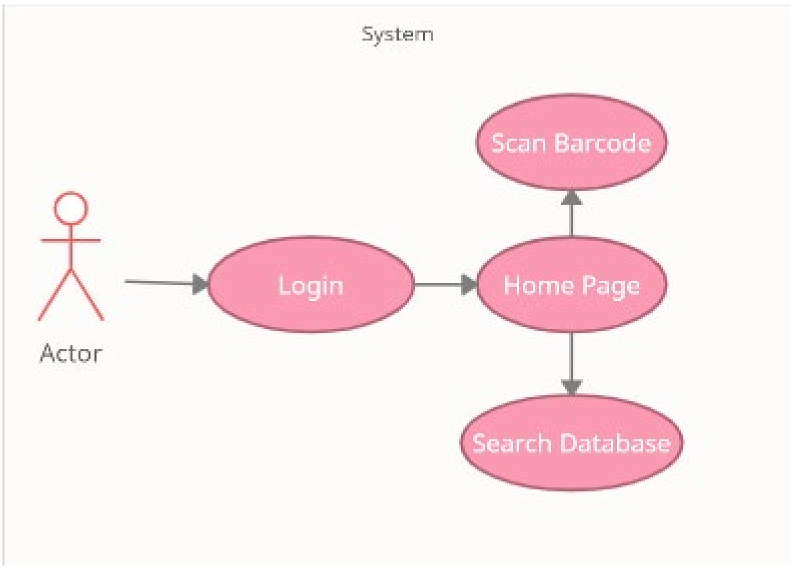


Figure 2 Use case diagram for employee searching stock room requirement

Flow Description

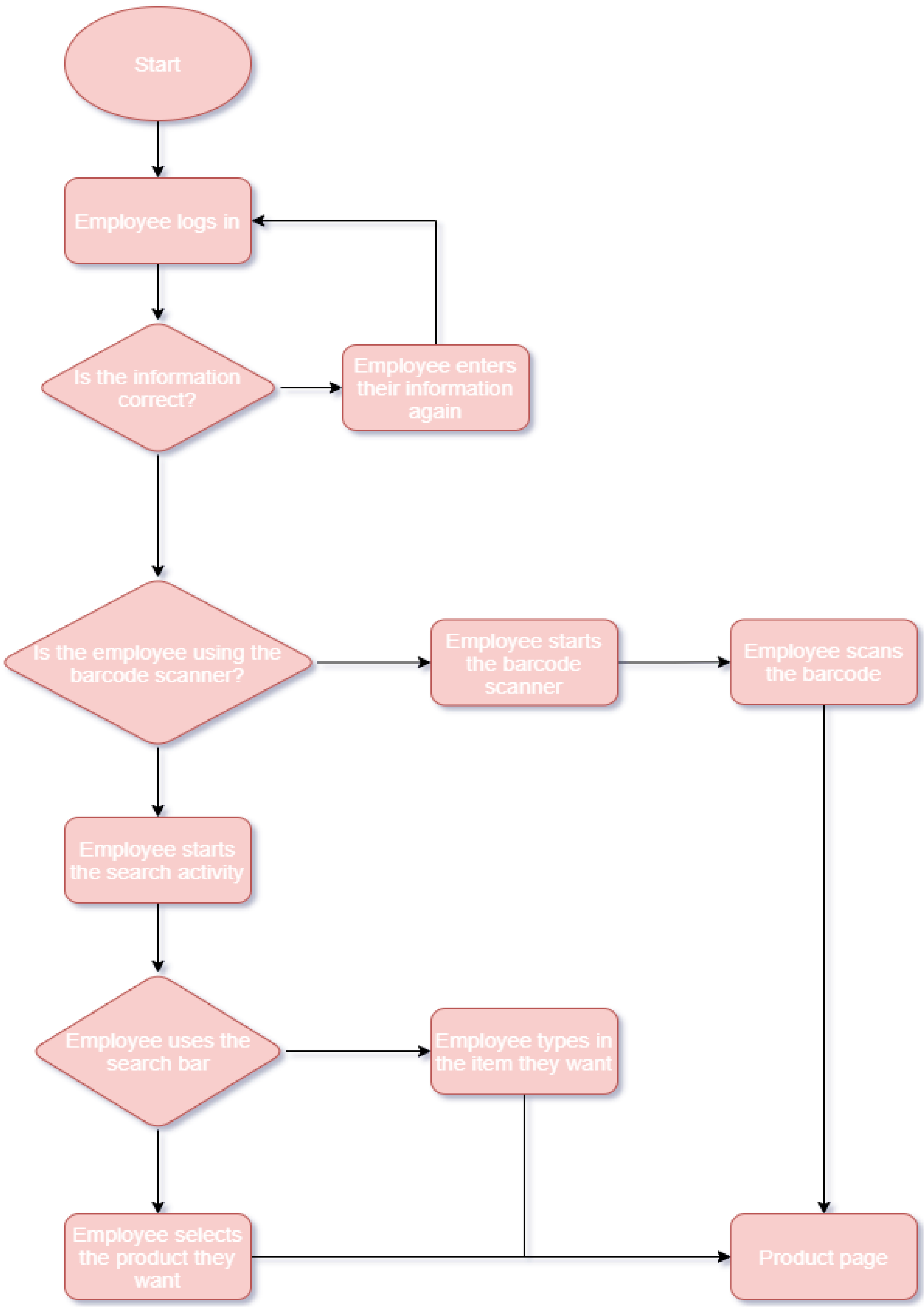

Start
Employee logs in
Is the information correct?
Employee enters their information again
Is the employee using the barcode scanner?
Employee starts the barcode scanner
Employee scans the barcode
Employee starts the search activity
Employee uses the search bar
Employee types in the item they want
Employee selects the product they want
Product page

Figure 3 Flow diagram for employee searching stock room requirement

Precondition
The application has been launched and the user has signed into their account.

Activation
This use case starts when a user clicks the search bar to search for an item in the database or starts the scan activity to scan a barcode which will return the item from the database.

Main flow
1. The system identifies the user wants to log in
2. The user types in their log in information and hits log in
3. The system verifies the information and logs in the user
4. The user clicks the search bar to search for an item
5. The system returns relevant results to the user

Alternate flow
1. The system identifies the user wants to log in
2. The user types in their log in information and hits log in
3. The system verifies the information and logs in the user
4. The user clicks the scan button
5. The system opens up the barcode scanning activity
6. The user scans a barcode and is shown matches for the barcode in the database

Termination
When the results are returned to the user

Post condition
The user is presented with the results of their search with an individual view of the product with more information on it.

2.1.1.3 Requirement 2: Employee creating a sale

2.1.1.3.1 Description & Priority

The requirement is about the employee adding an item to the cart of the system. This requirement has a priority of high due to it being a fundamental aspect of the system.

2.1.1.3.2 Use Case

Scope

The scope of this use case is to show an employee adding an item to the cart system.

Description

This use case describes the process in which an employee logs into the application and either adds an item to the cart by either scanning a barcode and adding from there or searching it the database from there and adding to the cart

Use Case Diagram

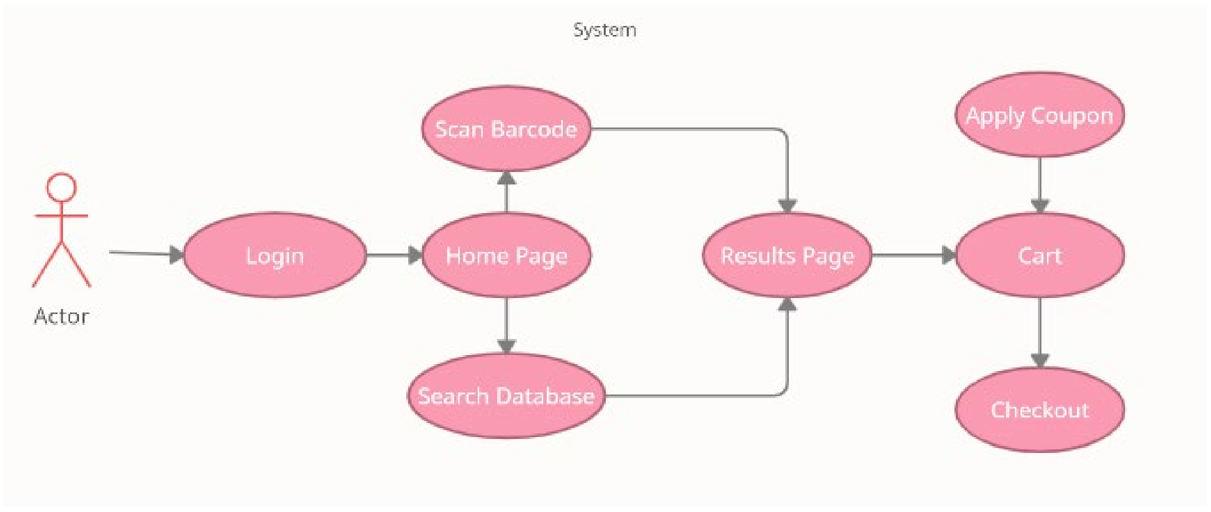


Figure 4 Use case diagram for employee creating a sale requirement

Flow Description

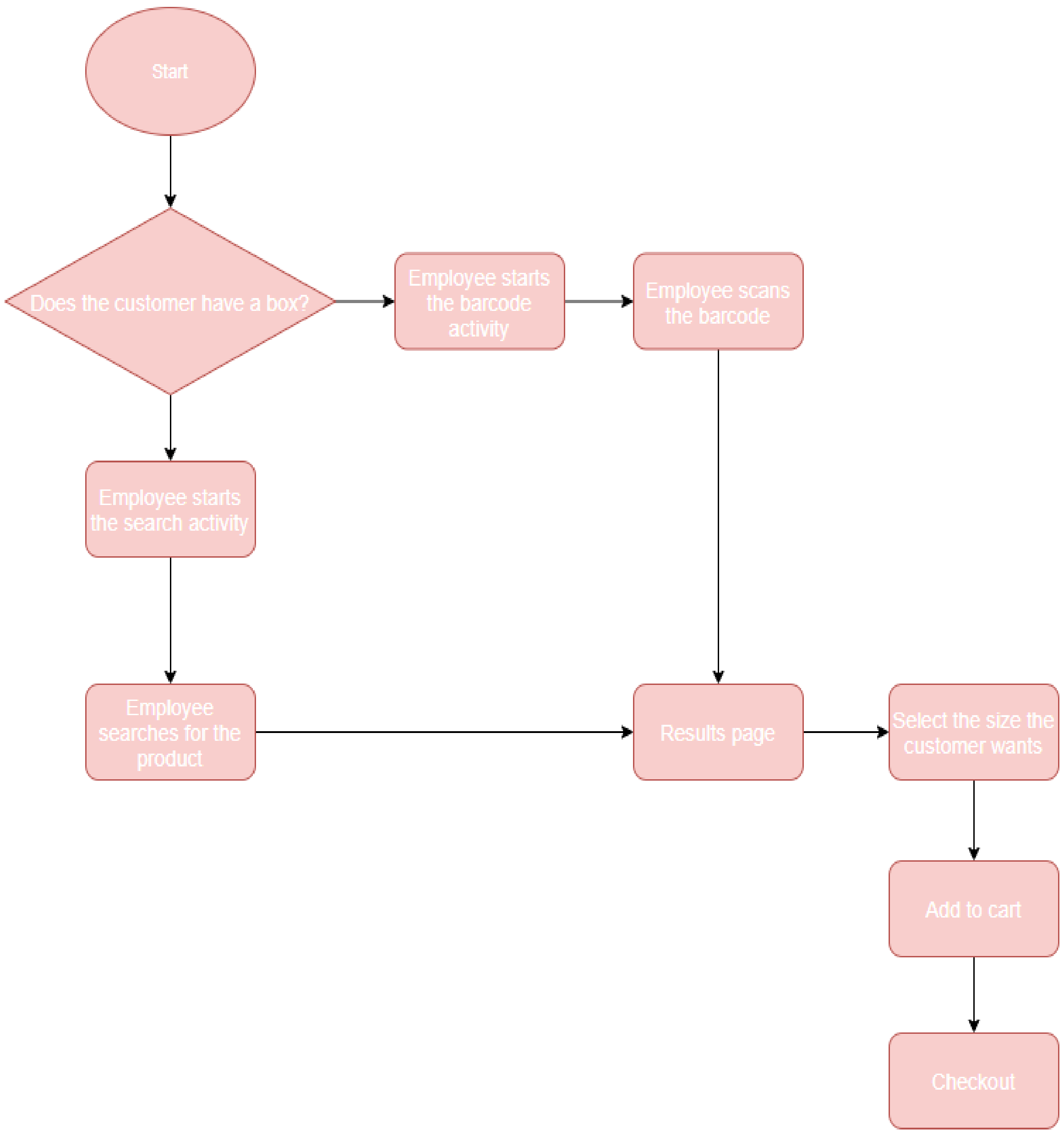

Start
Does the customer have a box?
Employee starts the barcode activity
Employee scans the barcode
Employee starts the search activity
Employee searches for the product
Results page
Select the size the customer wants
Add to cart
Checkout

Figure 5 Flow diagram for employee creating a sale requirement

Precondition

The application has been launched and the user has signed into their account and either scanned a barcode or searched for an item in the database.

Activation

This use case starts when a user clicks add item to the cart.

Main flow

1. The system returns relevant results to the user
2. The user clicks into the item they want to view
3. The user clicks add item to cart after selecting the size they want
4. The item is added to the cart

Alternate flow

1. The system returns relevant results to the user
2. The user clicks into the item they want to view
3. The user clicks add item to cart without selecting a size
4. The system informs the user that a size must be selected
5. The user selects a size and tries to add the item to the cart
6. The system allows this since a size has been selected

Termination

When the item is added to the cart.

Post condition

There is an item in the cart.

2.1.1.4 Requirement 3: Employee cancelling a sale

2.1.1.4.1 Description & Priority

The requirement is about the employee removing an item from the cart. Due to the cart being a fundamental aspect of my application working I am assigning it a priority of high

2.1.1.4.2 Use Case

Scope

The scope of this use case is to show an employee removing an item from the cart.

Description

This use case describes the process in which an item has been added to the cart but is no longer wanted by the customer, so it needs to be removed.

Use Case Diagram

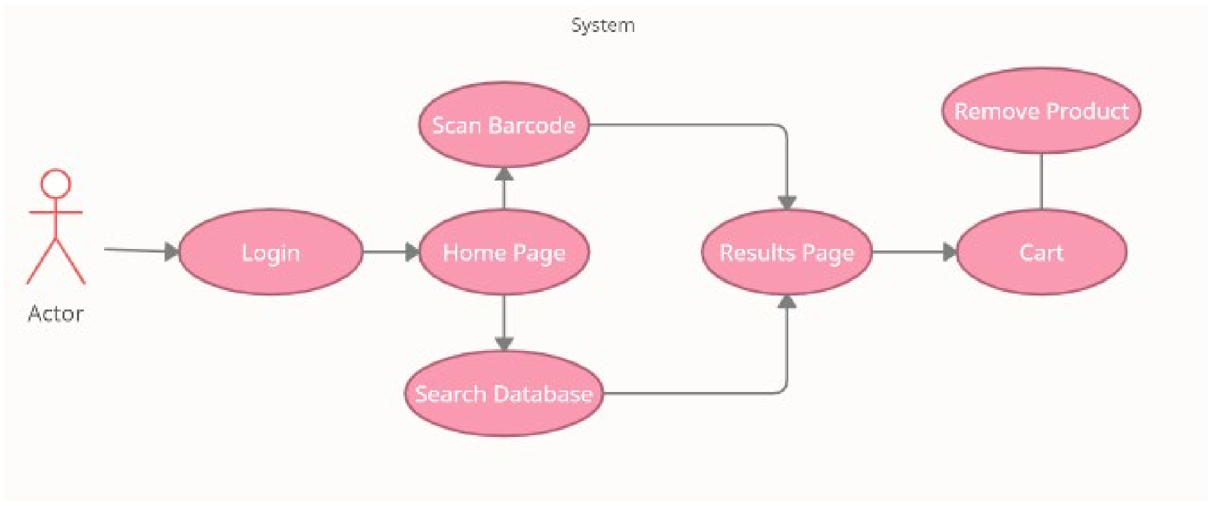


Figure 6 Use case diagram for employee cancelling a sale requirement

Flow Description

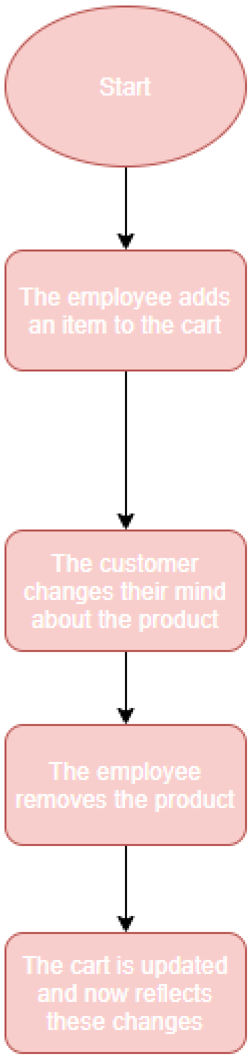

Start
The employee adds an item to the cart
The customer changes their mind about the product
The employee removes the product
The cart is updated and now reflects these changes

Figure 7 Flow diagram for employee cancelling a sale requirement

Precondition

There is an item in the cart.

Activation

This use case starts when the employee clicks the remove button on an item in the cart.

Main flow

1. The system removes the item from the cart
2. The stock levels for the item are readjusted
3. The total price of the cart is updated

Alternate flow

Termination

When the item is removed from the cart.

Post condition

The item that was selected was removed from the cart. The total price reflects this change.

2.1.1.5 Requirement 4: Employee applying a discount

2.1.1.5.1 Description & Priority

This requirement is about an employee applying a discount to the cart if a customer has a coupon. While this requirement is important it is not essential to the system working and so I am giving it a priority of medium.

2.1.1.5.2 Use Case

Scope

The scope of this use case is to show an employee applying a discount to the cart

Description

This use case describes the process in which an item or items have been added to the cart and the customer wishes to apply a discount to the cart.

Use Case Diagram

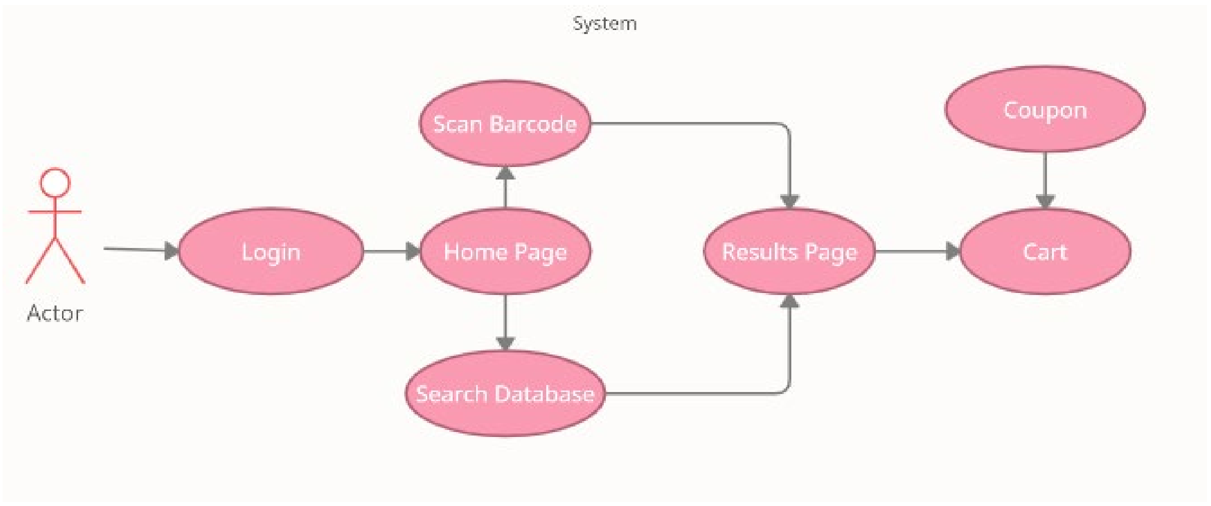


Figure 8 Use case diagram for employee applying a discount requirement

Flow Description

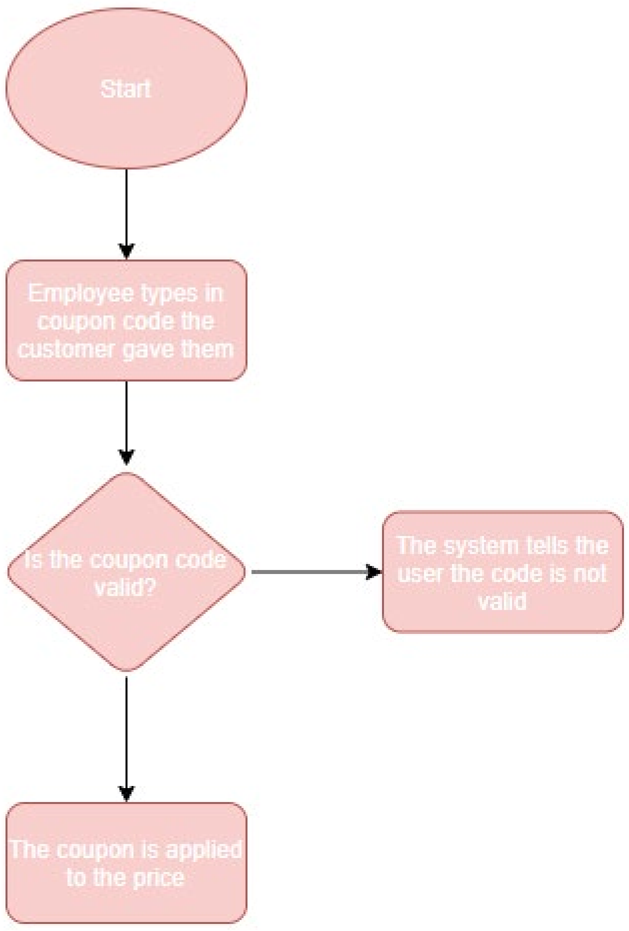

Start
Employee types in coupon code the customer gave them
Is the coupon code valid?
The system tells the user the code is not valid
The coupon is applied to the price

Figure 9 Flow diagram for employee applying a discount

Precondition

There is an item or items in the cart.

Activation

This use case starts when the employee clicks into the applying discount box.

Main flow

1. The system checks if the discount code is valid.
2. If the code is valid the system updates the cost of the cart.
3. The amount discounted is displayed above the total amount.

Alternate flow

Termination

The user clicks the apply discount button.

Post condition

The discount has been applied.

2.1.1.6 Requirement 5: Customer paying

2.1.1.6.1 Description & Priority

This requirement is about a customer paying for the items they have selected in the store. This requirement has a priority of high due to it being a fundamental aspect of the system.

2.1.1.6.2 Use Case

Scope

The scope of this use case is to show a customer paying for the items in their checkout.

Description

This use case describes a customer paying for the items they have selected and are in the cart.

Use Case Diagram

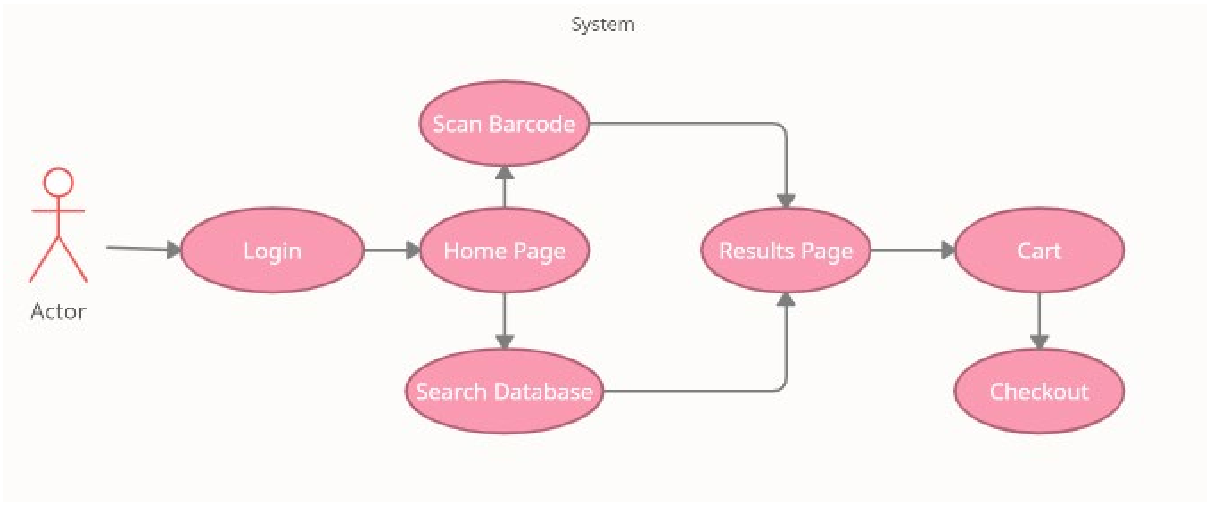


Figure 10 Use case diagram for customer paying requirement

Flow Description

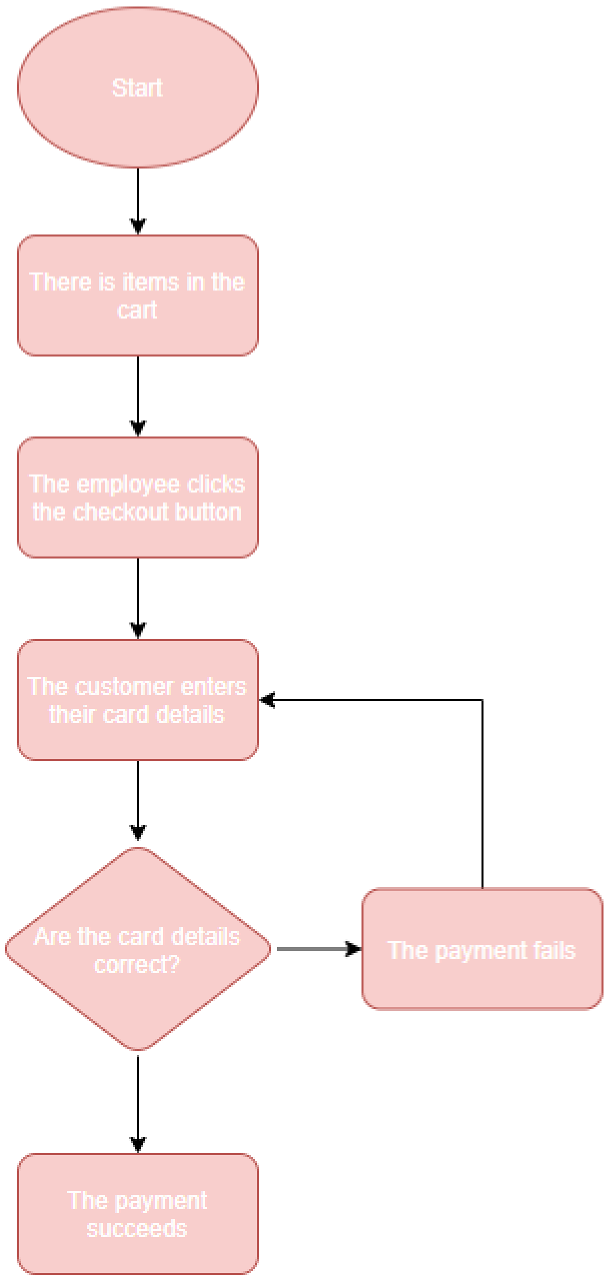

Start
There is items in the cart
The employee clicks the checkout button
The customer enters their card details
Are the card details correct?
The payment fails
The payment succeeds

Figure 11 Flow diagram for customer paying requirement

Precondition

There is an item or items in the cart.

Activation

This use case starts when the employee clicks the checkout button.

Main flow

1. The system confirms the items in the cart and the total price.
2. The system opens up the checkout activity and prompts the customer to pay.
3. The system validates the payment.

Alternate flow

1. The system confirms the items in the cart and the total price.
2. The system opens up the checkout activity and prompts the customer to pay.
3. The customer enters their card details and clicks pay.
4. The payment fails and the customer is allowed try again if they want

Termination

The user clicks the back button to cancel the order.

Post condition

The user is brought to the home activity with a message letting them know the order has been completed.

2.1.1.7 Requirement 6: Refunding a customer

2.1.1.7.1 Description & Priority

This requirement is about giving a customer a refund if they are not happy with the products they have received. Being able to refund a customer is essential to the operations of a retail store and so the priority of this requirement is high.

2.1.1.7.2 Use Case

Scope

The scope of this use case is to show a customer getting a refund for their entire order.

Description

This use case describes an employee issuing a refund to a customer for the entirety of their order.

Use Case Diagram

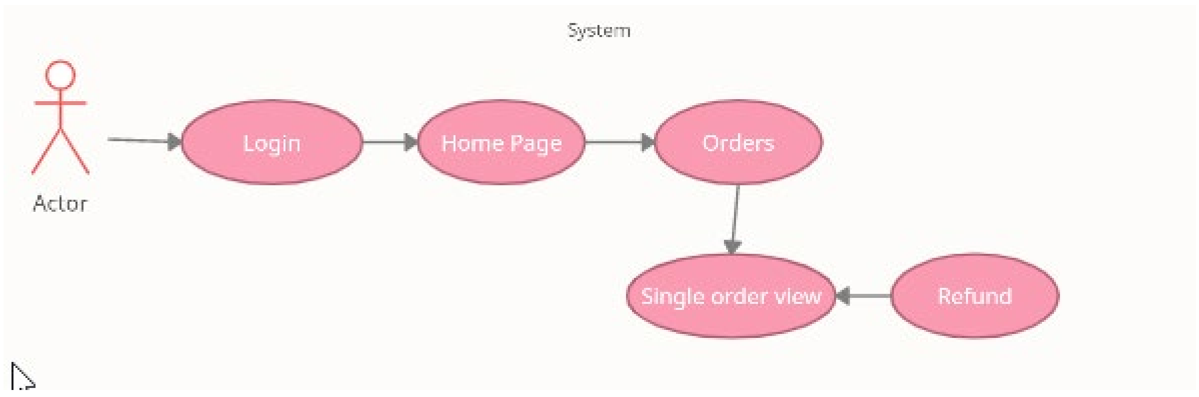


Figure 12 Use diagram for refunding a customer requirement

Flow Description

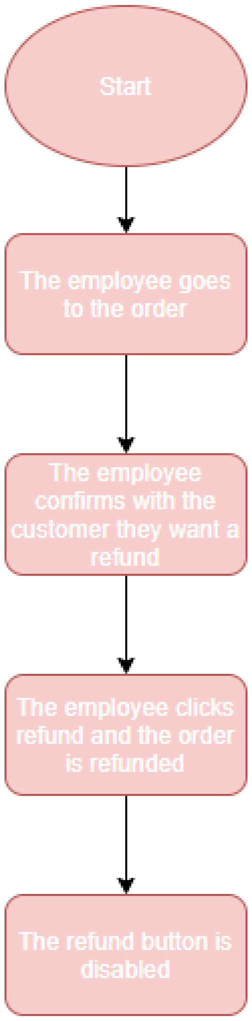

Start
The employee goes to the order
The employee confirms with the customer they want a refund
The employee clicks refund and the order is refunded
The refund button is disabled

Figure 13 Flow diagram for refunding a customer requirement

Precondition

The employee dealt with the customer when completing their order before and the order went through successfully.

Activation

This use case starts when the employee is in the order view activity and clicks into the specific order that needs to be refunded.

Main flow

1. The system sends a request to the Node.js server
2. The Node.js server sends a request to stripes API.
3. Stripe responds to the Node.js server letting them know the refund was successful.
4. The Node.js server sends a response to the system and displays the toast message while disabling the button.

Alternate flow

Termination

The refund button is disabled, and the order has been refunded.

Post condition

The user is presented with a message letting them know that the refund was successful, and the refund button is disabled so no more refunds can be issued on that order.

2.1.1.8 Requirement 7: Registering

2.1.1.8.1 Description & Priority

This requirement is about allowing an employee to register their account in order to user the system. As a lot of the functionality of the application relies on a user being logged in this requirement is getting a priority of high.

2.1.1.8.2 Use Case

Scope

The scope of this use case shows a user registering their account in order to use the system.

Description

This use case describes the process an employee goes through in order to register an account with the system.

Use Case Diagram

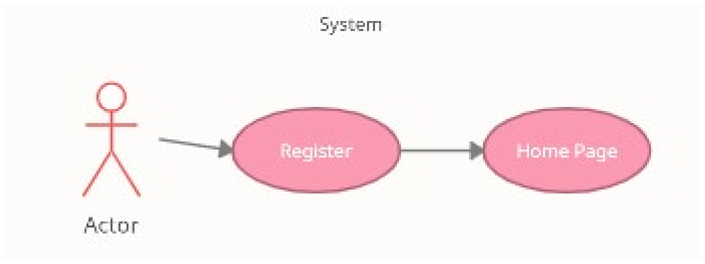


Figure 14 Use case diagram for registering requirement

Flow Description

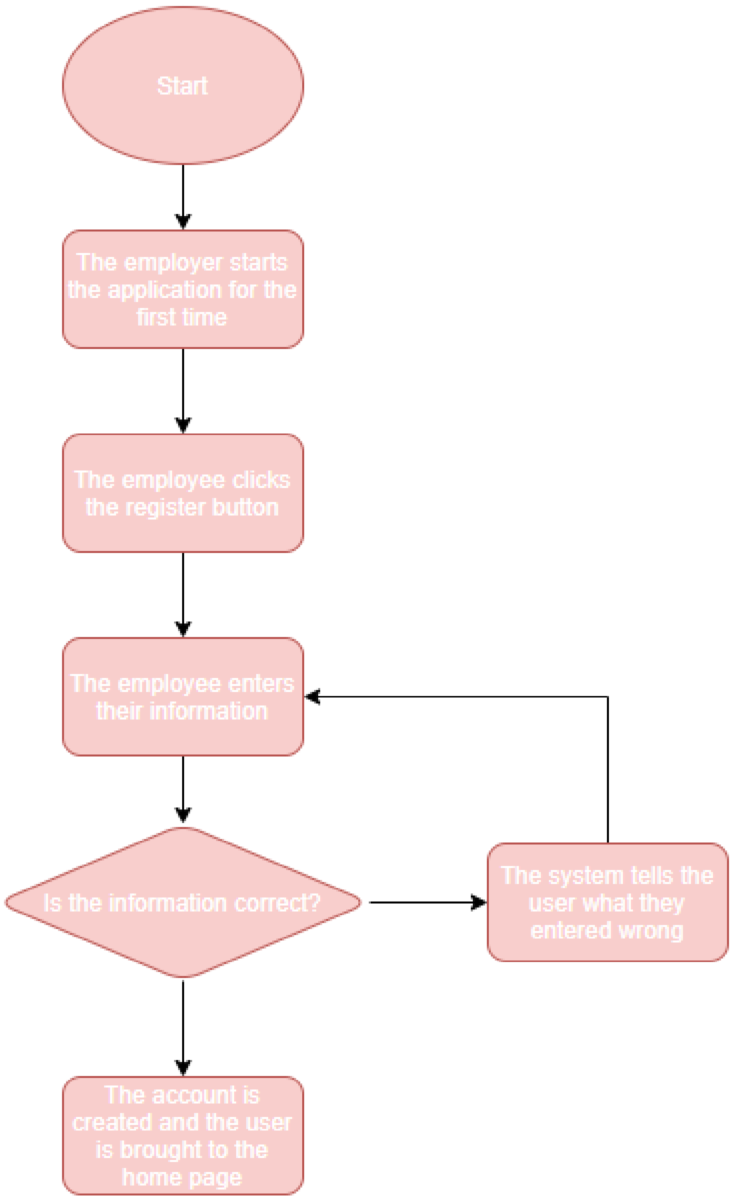

Start
The employer starts the application for the first time
The employee clicks the register button
The employee enters their information
Is the information correct?
The system tells the user what they entered wrong
The account is created and the user is brought to the home page

Figure 15 Flow diagram for registering requirement

Precondition

The employee has the device downloaded on their device and has opened the application.

Activation

This use case starts when an employee starts the application and clicks the register an account button on the home activity.

Main flow

1. The employee opens up the application and asks to register an account.
2. They are brought to the register activity and fill in their information.
3. The employee clicks register after filling in all their information and a request is sent to the firebase database to make a user.
4. The user is created, and the employee is brought to the main page of the activity.

Alternate flow

1. The employee opens up the application and asks to register an account.
2. They are brought to the register activity and fill in their information.
3. The employee clicks register but they have not entered a valid email.
4. The employee is told this and asked to enter a valid email before trying to create an account.
5. The employee notices the spelling mistake in their email and fixes and clicks register again.
6. The user is created, and the employee is brought to the main page of the activity.

Termination

The employee is brought to the main page of the system giving him full access to all functionalities.

Post condition

The employee can now access all functions on the system and do anything his employer would need him to do.

2.1.1.9 Requirement 8: Logging in

2.1.1.9.1 Description & Priority

This requirement is about allowing an employee to login once they have registered an account. It will allow the employee to view their past orders as well as their current cart. This requirement has a priority of high.

2.1.1.9.2 Use Case

Scope

The scope of this use case is to show an employee logging into the system and any errors they may encounter when doing this.

Description

This use case describes an employee logging in with an account they have already created themselves.

Use Case Diagram

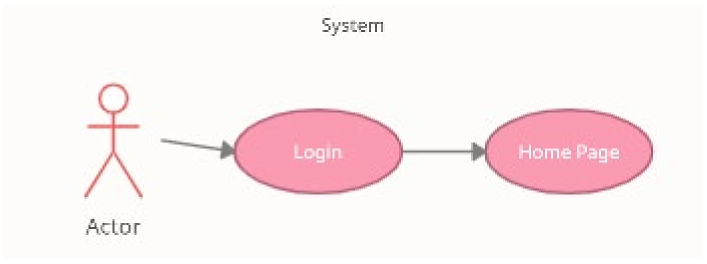


Figure 16 Use case diagram for logging in requirement

Flow Description

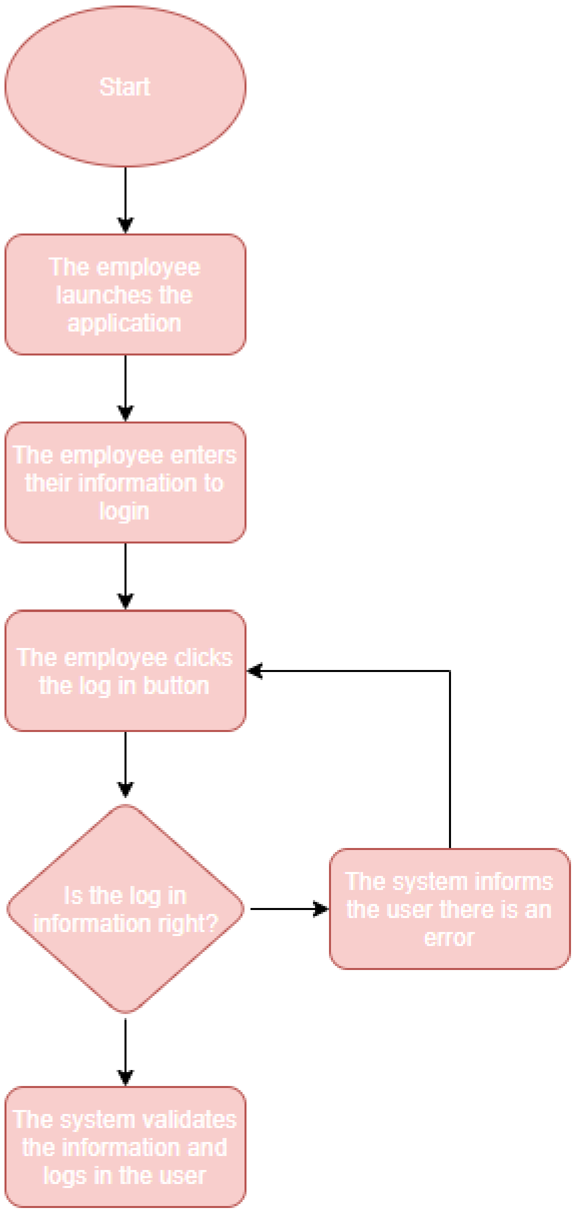

Start
The employee launches the application
The employee enters their information to login
The employee clicks the log in button
Is the log in information right?
The system informs the user there is an error
The system validates the information and logs in the user

Figure 17 Flow diagram for logging in requirement

Precondition

The employee has an account on the system and knows the information required to log in to the account.

Activation

This use case starts when the employee opens the application and is asked to log in.

Main flow

1. The employee enters their information into the required fields and clicks log in.
2. A request is sent to the firebase database to authenticate their information.
3. Firebase sends back a successful authentication message and allows the user to log in.

Alternate flow

1. The employee enters their information into the required fields and clicks log in.
2. A request is sent to the firebase database to authenticate their information.
3. Firebase sends back a failed authentication message and informs the employee what the error is.
4. The employee fixes their mistake and clicks log in again
5. A new request is sent to the firebase database to authenticate their information and it responds back with a successful authentication message.

Termination

The employee is on the home screen activity of the system.

Post condition

The employee can now access all functions on the system and do anything his employer would need him to do.

2.1.1.10 Requirement 9: Logging out

2.1.1.10.1 Description & Priority

This requirement is about allowing a user to log out of the system once they are done using it. This is standard functionality of an application but does not necessarily impact any other functionality on the application. For this reason, it is given a priority of medium.

2.1.1.10.2 Use Case

Scope

The scope of this use case is to show an employee logging out of the system.

Description

This use case describes an employee logging out of the system and the impacts that has on the system.

Use Case Diagram

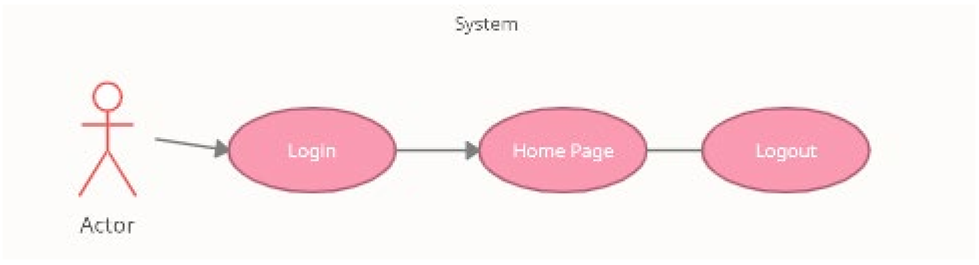


Figure 18 Use case diagram for logging out requirement

Flow Description

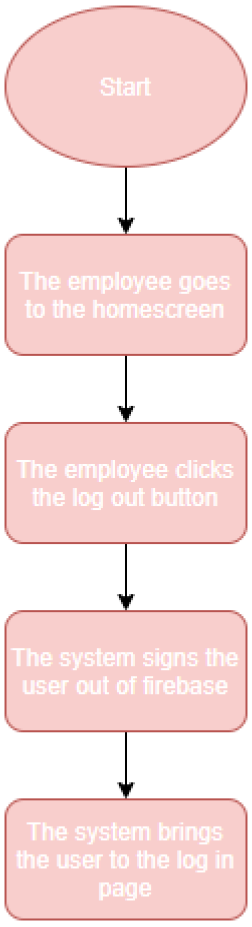

Start
The employee goes to the homescreen
The employee clicks the log out button
The system signs the user out of firebase
The system brings the user to the log in page

Figure 19 Flow diagram for logging out requirement

Precondition

The employee has logged into the system and is on the home page of the system.

Activation

This use case starts when the employee clicks the log out button.

Main flow

1. The system signs the user out of the firebase database.
2. The System logs the user out and brings them to the login activity.

Alternate flow

Termination

The user is on the log in page activity.

Post condition

The employee has no access to the systems functionality.

2.1.2 Data Requirements

2.1.2.1 Requirement 1: Access to the database

In order for my application work as intended the user needs access to the connected google firebase database. The functions of the application will still work however they will not be able to return any results if the user is not connected to the firebase database.

2.1.3 User Requirements

2.1.3.1 Requirement 1: Access to the internet

Expanding on the data requirement of having access to the database, for a user to fulfil that requirement they will need to have a device with internet connectivity.

2.1.3.2 Requirement 2: Access to the camera

In order for the system to work fully the user needs to give the system permission to use the devices camera for the barcode scanner.

2.1.2.3 Requirement 2: Access to the Node.js server

In order for the system to issue refunds and facilitate payments it must have a connection to the Node.js server.

2.1.3 Environmental Requirements

2.1.4 Usability Requirements

2.1.4.1 Requirement 1: Ability to use the system without error

With this requirement the aim is to ensure that an end user can use the system with minimal errors after a week of using the application while at work.

2.2 Design & Architecture

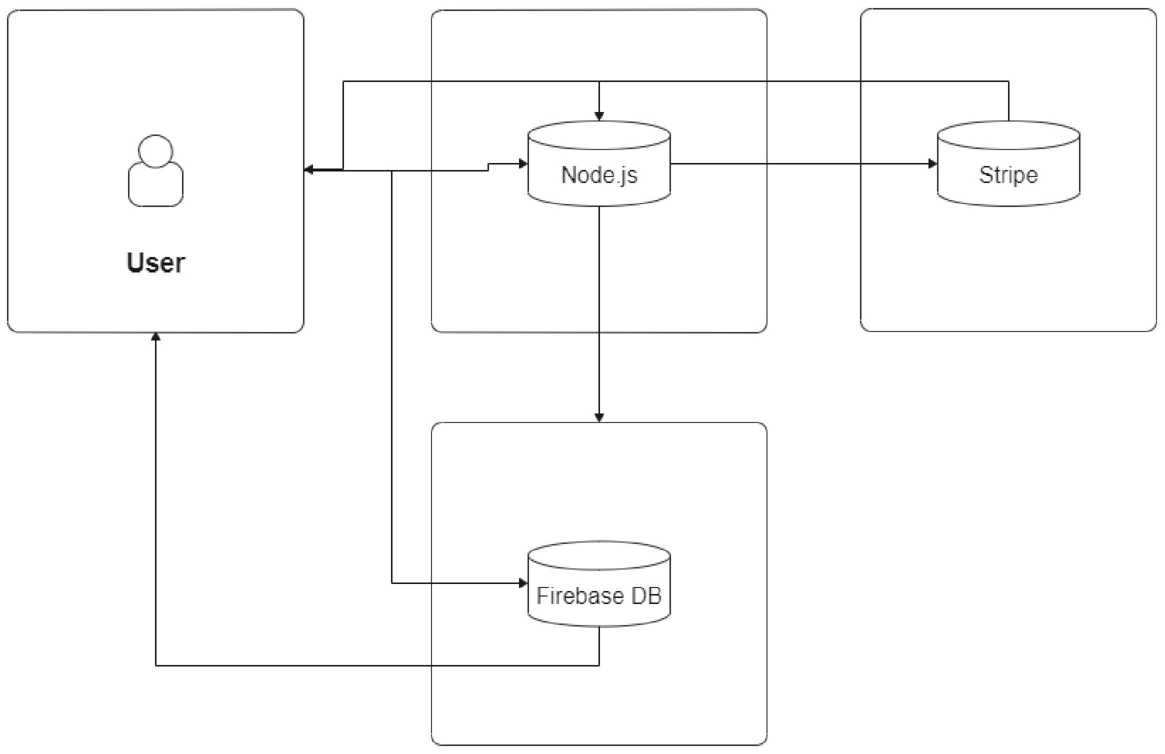


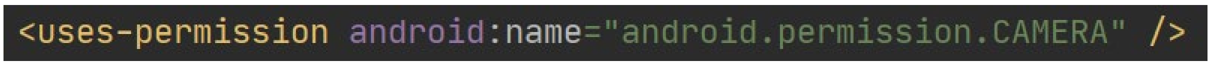
Figure 20 System architecture diagram

From this system architecture diagram, you can see there are several components to my system. The first being the firebase database, one of two external servers that make up the system. All of the product data and employee data is stored on this database. The second component is the Node.js server that acts as a middleman between the mobile application and stripes API. The Node.js server is required in order to facilitate both refunds and payments with the logic for those functionalities being stored on that server. The last external component to the system is Stripes API which we send our refund and payment requests to. Finally, there is the mobile application which the end user interacts with as a front-end side to the system.

2.3 Implementation

One of the fundamental requirements of the project was to allow the scanning of barcodes. This is because barcodes are a fundamental way of creating a sale in a retail store. In this section I will be going over how I have implemented my barcode scanner.

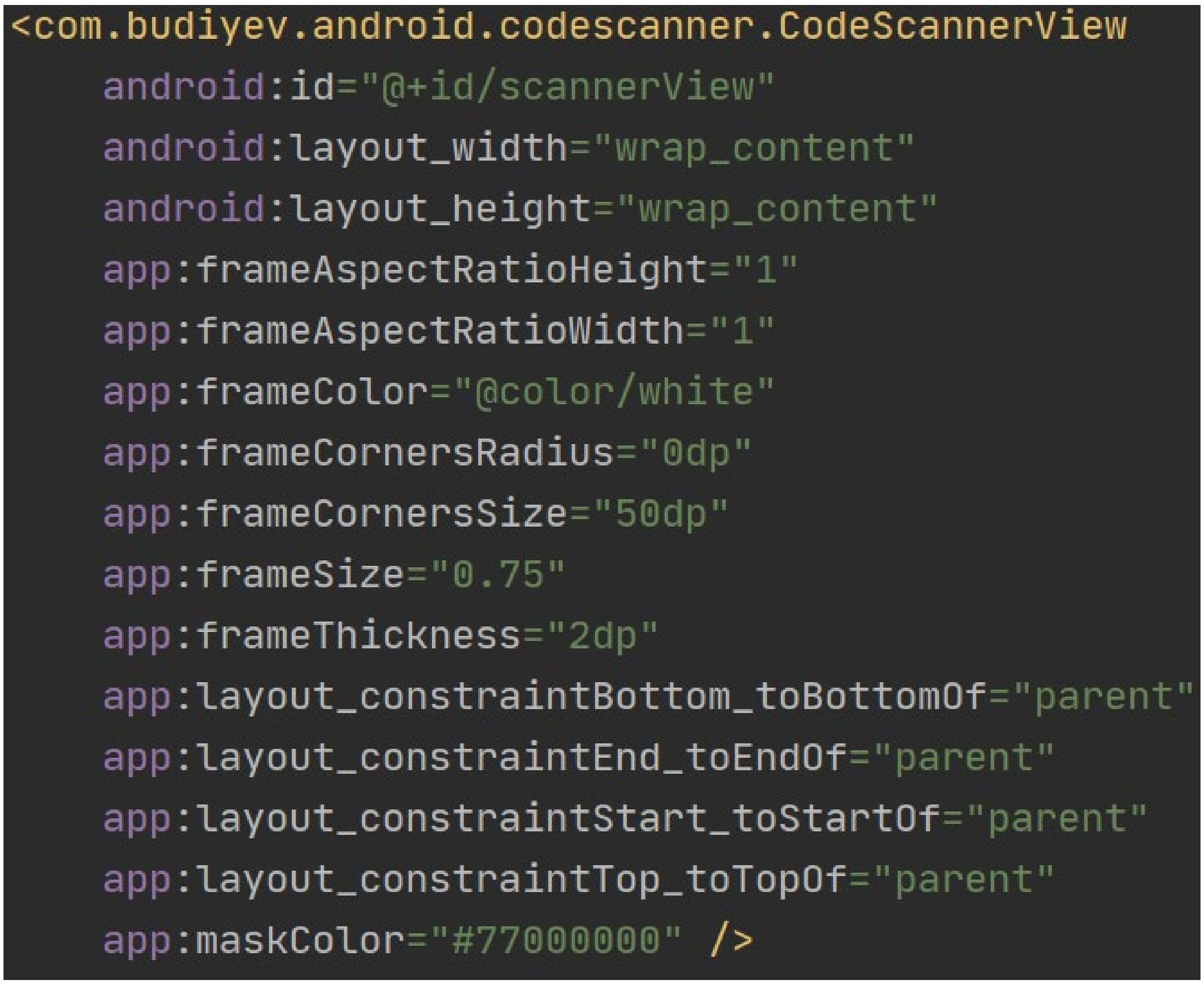
The first step in using a barcode scanner in my system was finding a library that facilitated the functionality. After doing some research I settled on a barcode scanner created by a GitHub user by the name of Yury-Budiyev. I chose this library because of its simplicity and its ability to do the functionality I required. With my library being selected the next step was to get it working within my project. This involved getting permission to use the devices camera when the application is open. In order to do this, we have to create a uses-permission tag in the manifest of the project.



```
<uses-permission android:name="android.permission.CAMERA" />
```

Figure 221 Requesting camera permission inside AndroidManifest.xml

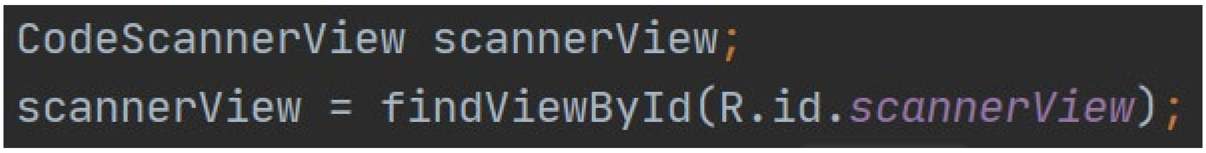
The next step in this process is to create a view of what the camera sees so the employee can see this also. Without this step when the user starts the activity, they would only see a black screen and would not be able to see barcodes. This is done by using a widget belonging to the scanner library that we place on the relative xml layout.



```
<com.budiyev.android.codescanner.CodeScannerView
    android:id="@+id/scannerView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:frameAspectRatioHeight="1"
    app:frameAspectRatioWidth="1"
    app:frameColor="@color/white"
    app:frameCornersRadius="0dp"
    app:frameCornersSize="50dp"
    app:frameSize="0.75"
    app:frameThickness="2dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:maskColor="#77000000" />
```

Figure 222 Layout of the scanner view inside activity_scanner.xml

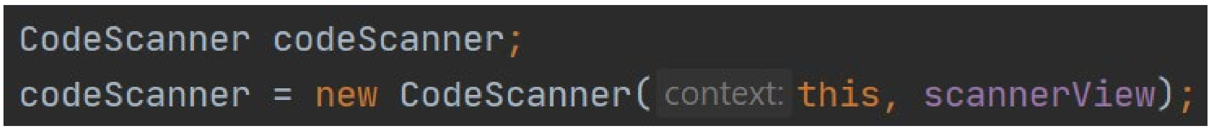
Now that when the employee starts the activity, they can see what the camera sees the next step is to allow the functionality of scanning barcodes. In order to do this, we need a Code Scanner object. To make this object we need the context of the system and a Scanner View object. The Scanner View object we have created above in the xml class and just need to instantiate.



```
CodeScannerView scannerView;
scannerView = findViewById(R.id.scannerView);
```

Figure 223 Instantiating the Scanner View Object inside Scanner.java

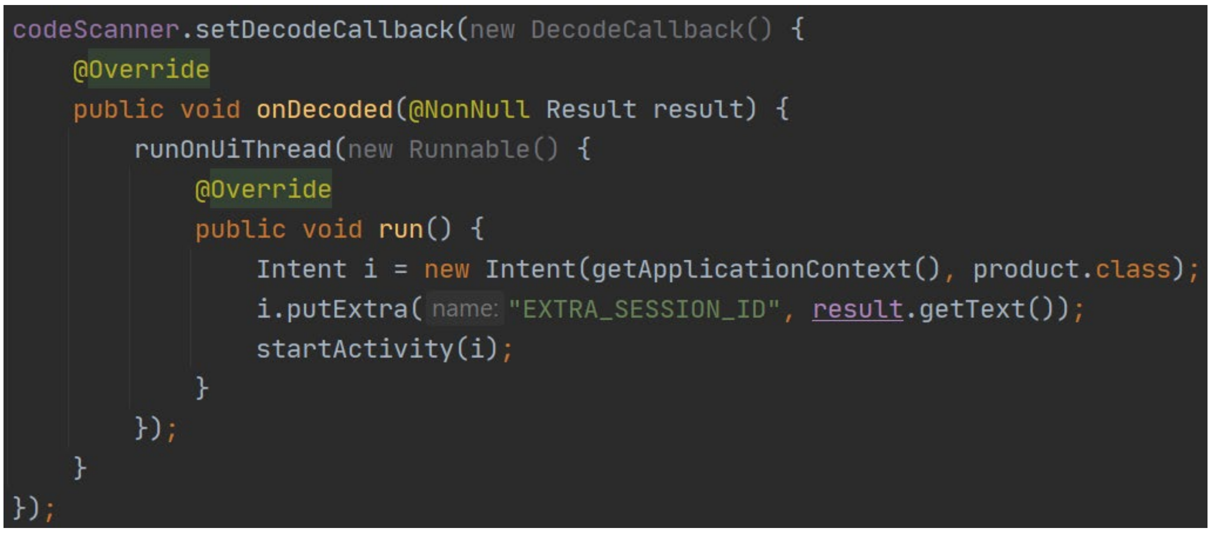
Now that we have everything needed to make the Code Scanner object the next step is to create that object and use it to scan barcodes. Like I said above we need two things to create this object. The Scanner View object and the context of the system. We pass those values into the objection and this now leaves us with the ability to scan barcodes.



```
CodeScanner codeScanner;
codeScanner = new CodeScanner( context: this, scannerView);
```

Figure 224 Creating the barcode Scanner object inside Scanner.java

So now that we have the object, we can scan the barcodes, and this is done using the setDecodeCallBack method from the library we are using. With this method it returns a result object that we can use to find the specific product we want. We can manipulate this result object into a string that we then can use to query the firebase database.



```
codeScanner.setDecodeCallback(new DecodeCallback() {
    @Override
    public void onDecoded(@NonNull Result result) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Intent i = new Intent(getApplicationContext(), product.class);
                i.putExtra( name: "EXTRA_SESSION_ID", result.getText());
                startActivity(i);
            }
        });
    }
});
```

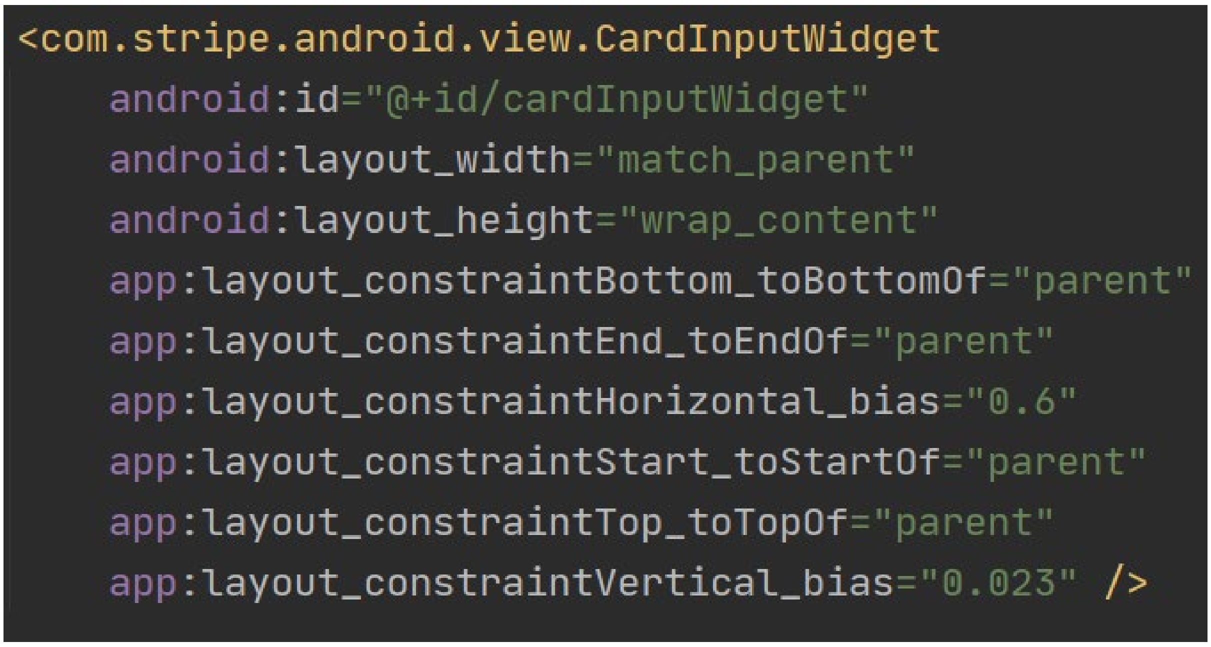
Figure 225 decoding a scanned barcode inside Scanner.java

You can see above that when we scan a barcode, we are starting the product class. This is a java class that will show an individual product for whatever ID is passed in. In my use case my products go by numbers from 01 to 10. So, when scanning a barcode with a value of 07 the result object will consist of only 07. This is what allows the system to match barcodes with products stored in the database.

Payment

One of the key aspects of the project is to facilitate payments. Without this functionality the project would not be viable in its desired environment. In the below section I will be going over how I have facilitated this functionality.

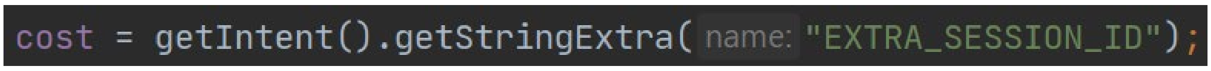
The first step in facilitating payments is finding the right company to handle my transaction. There are various companies that allow this functionality however I went with Stripe as they gave me the opportunity to work with a new technology, Node.js, and had a test API key that allowed me to do mock transactions. With this decided on I started on creating the layout the employee would see when they started the checkout activity. Stripe have a card input widget that does a lot of the heavy lifting of handling the card details. This is done mostly for secure reasons that ensure that when the request is sent to Stripe that the card details are encrypted.



```
<com.stripe.android.view.CardInputWidget
    android:id="@+id/cardInputWidget"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.6"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.023" />
```

Figure 226 Creating the card input widget inside activity_checkout.xml

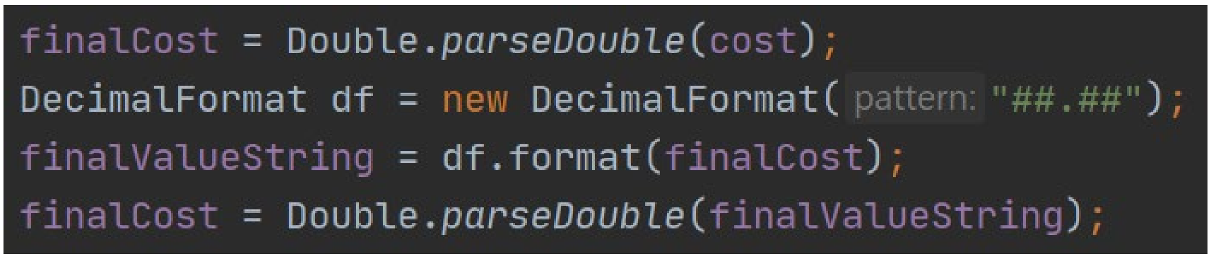
Now that the customer can enter their card details when needed we have to handle the backend side of things. In order for a customer to pay, there has to be an amount that is sent to the Node.js server. This amount is passed in from the cart activity where the total cost of all the products the customer wants is calculated.



```
cost = getIntent().getStringExtra( name: "EXTRA_SESSION_ID");
```

Figure 227 Getting the amount from the cart activity inside checkout.java

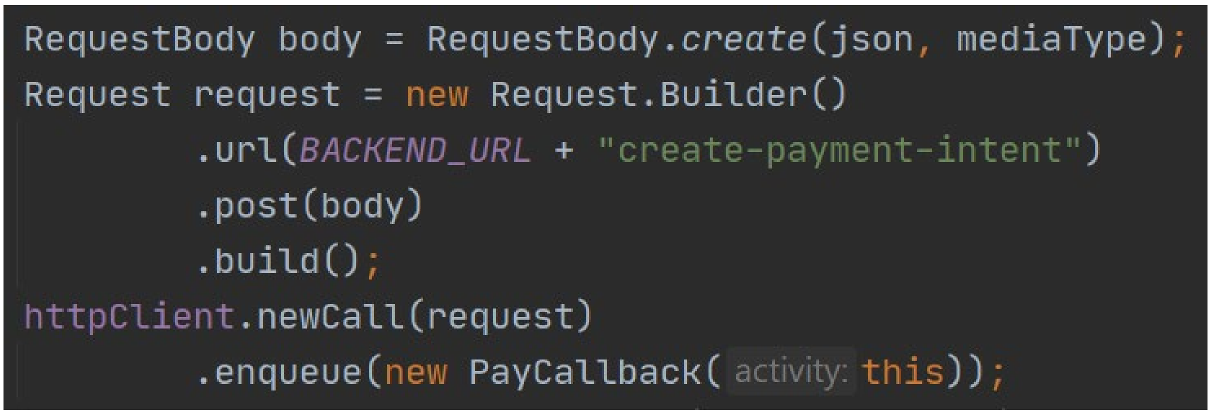
While we have the amount there is still a lot that needs to be done to it before it can be worked with. The first step we need to take is to convert it into a double for two reasons. The first being Stripe requires the amount to be passed over as a double and the second being we need to do some formatting on the value. Due to the use of coupons the amount passed over can be several decimal places long. However, Stripe does not like this, and it causes payments to fail. With this in mind we format the newly created double to round to two decimal places. However, in order to do the formatting, it must be converted back into a string. Note you cannot format a string and that is why it is converted to a double initially. Finally, we convert back to a double and we have our amount in the right format needed.



```
finalCost = Double.parseDouble(cost);
DecimalFormat df = new DecimalFormat( pattern: "##.##");
finalValueString = df.format(finalCost);
finalCost = Double.parseDouble(finalValueString);
```

Figure 228 Formatting the amount to ensure Stripes system accepts it

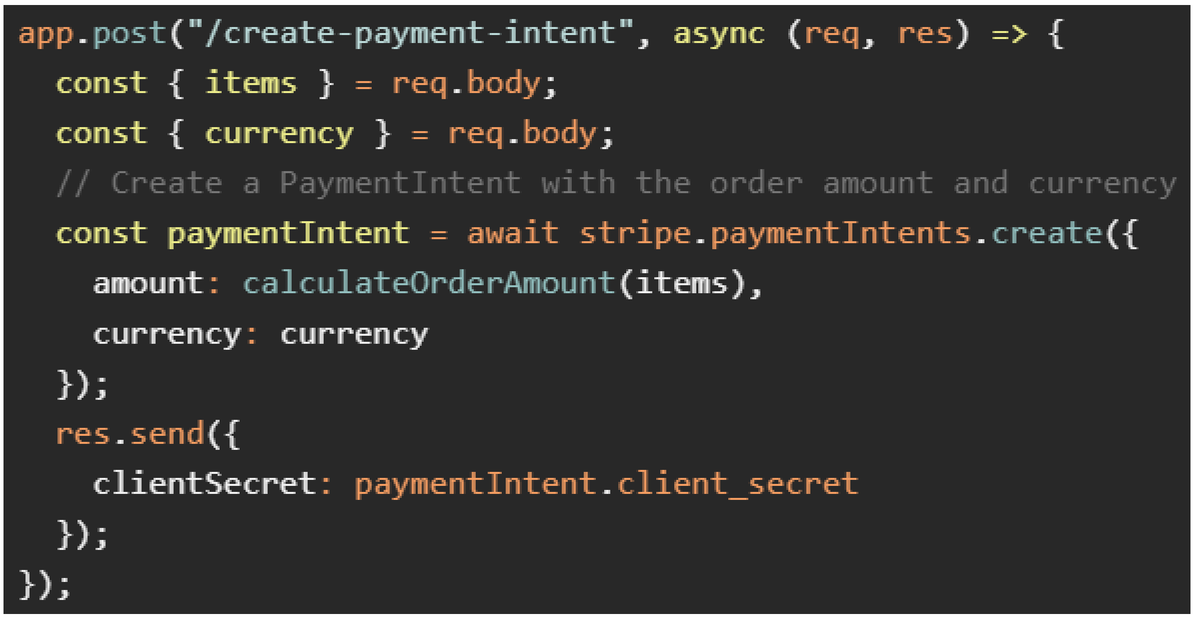
Now that we have the amount, we can make our request to the Node.js server. This is done by creating a http request consisting of a json object. Inside this json object there is the amount of the transaction and the currency to do the transaction in.



```
RequestBody body = RequestBody.create(json, mediaType);
Request request = new Request.Builder()
        .url(BACKEND_URL + "create-payment-intent")
        .post(body)
        .build();
httpClient.newCall(request)
        .enqueue(new PayCallback( activity: this));
```

Figure 229 creating a request and sending it to the Node.js server

Looking at the Node.js aspect of a payment we get the amount and currency from the json object we sent in the above request. With this information now on the Node.js server we can send a request to Stripe to ask them to facilitate the payment with those two parameters.

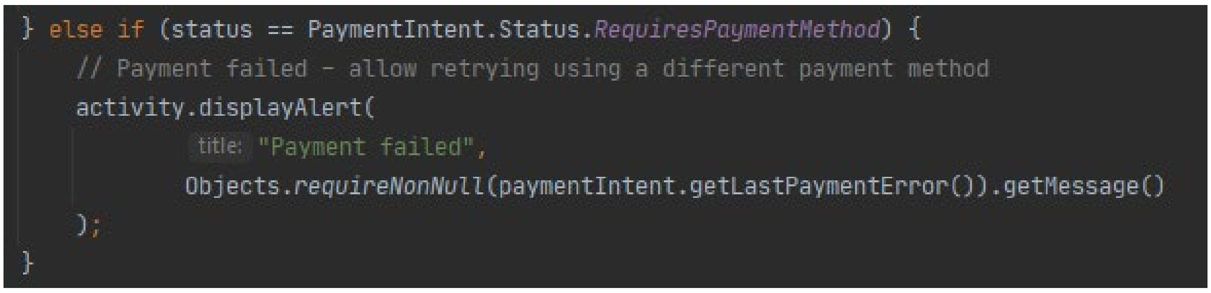


```
app.post("/create-payment-intent", async (req, res) => {
  const { items } = req.body;
  const { currency } = req.body;
  // Create a PaymentIntent with the order amount and currency
  const paymentIntent = await stripe.paymentIntents.create({
    amount: calculateOrderAmount(items),
    currency: currency
  });
  res.send({
    clientSecret: paymentIntent.client_secret
  });
});
```

Figure 30 Node.js server receiving the system request and sending a request to Stripe

You can also see that the method sends a response back to the system. This response consists of a success or failure value as well as a json object. The json object contains a plethora of information in regard to the transaction. Depending on if the transaction was a success or not one of two things can happen.

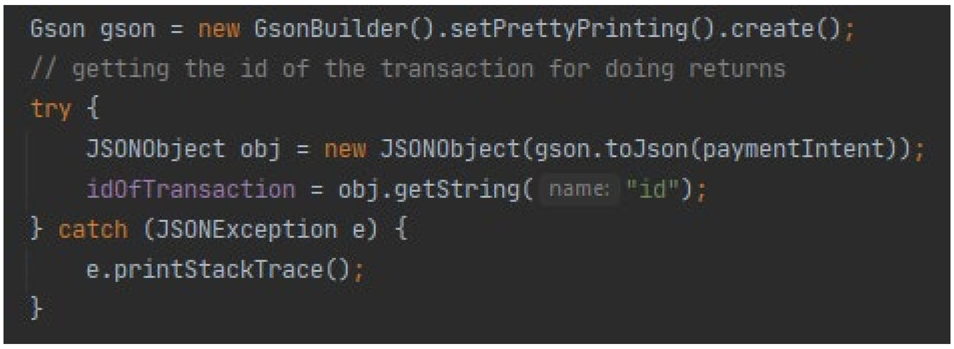
If the transaction failed to go through an alert is created informing the customer or employee whoever has the device at this point of the problem that happened. The user can then close that alert and try again if they wish.



```
} else if (status == PaymentIntent.Status.RequiresPaymentMethod) {
    // Payment failed - allow retrying using a different payment method
    activity.displayAlert(
            title: "Payment failed",
            Objects.requireNonNull(paymentIntent.getLastPaymentError()).getMessage()
    );
}
```

Figure 31 Payment failure

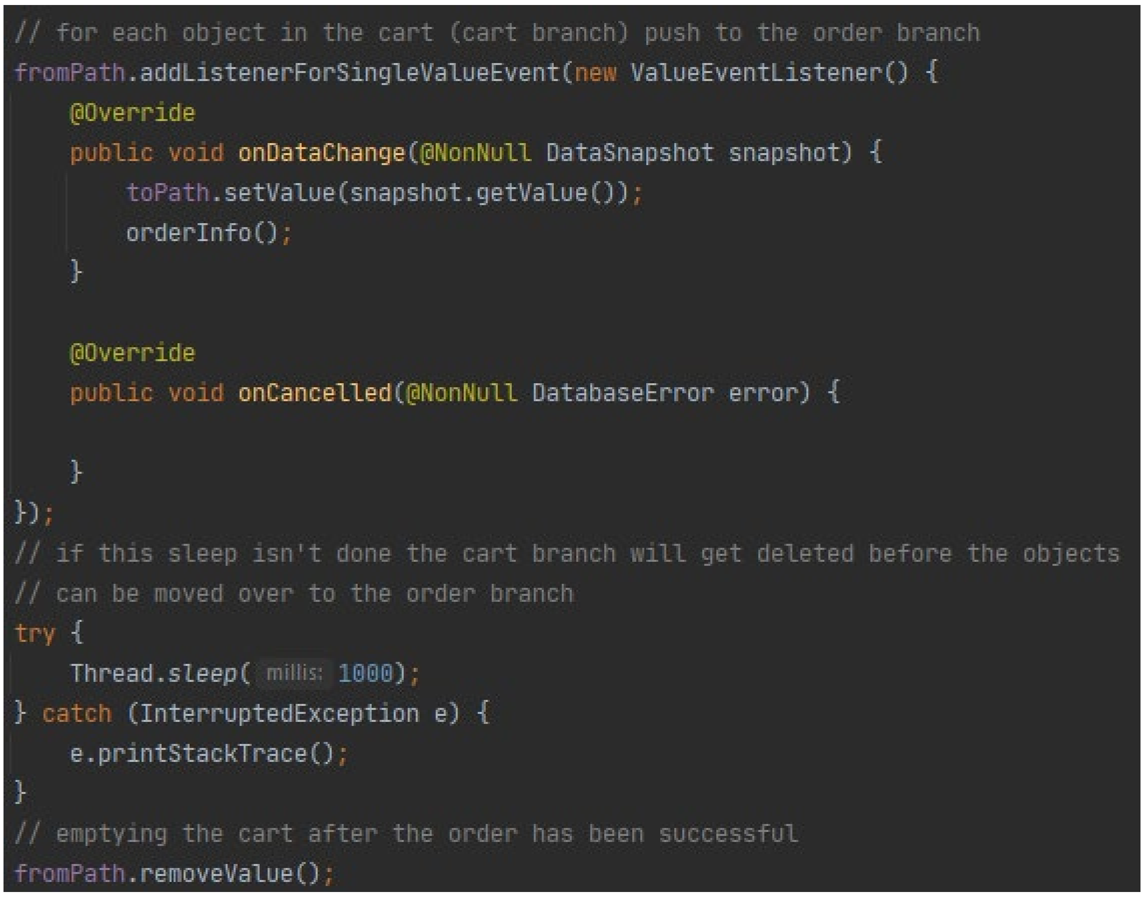
With the other option being if the payment succeeds. Three things happen in this instance. The first thing is capturing the transaction ID from the json object that is returned to us. This is done so that refunds can be done at a later date.



```java
Gson gson = new GsonBuilder().setPrettyPrinting().create();
// getting the id of the transaction for doing returns
try {
    JSONObject obj = new JSONObject(gson.toJson(paymentIntent));
    idOfTransaction = obj.getString( name: "id");
} catch (JSONException e) {
    e.printStackTrace();
}
```

Figure 32 Getting the transaction ID of the order

The second being a history of the order is created. This is done by making a query to the cart branch and moving all objects to a unique order branch. This unique value is the time the order was made. Once the order history has been created the cart is emptied for the next order. An order Info object is also created and pushed to the same branch of the database. This object contains nice to know information such a s the total cost of the order, if a coupon was used, the transaction id and if the order has been returned.



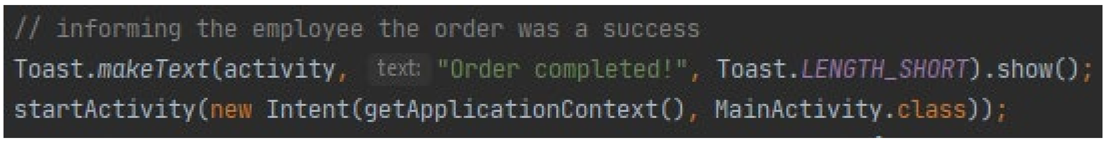
```java
// for each object in the cart (cart branch) push to the order branch
fromPath.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        toPath.setValue(snapshot.getValue());
        orderInfo();
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {

    }
});
// if this sleep isn't done the cart branch will get deleted before the objects
// can be moved over to the order branch
try {
    Thread.sleep( millis: 1000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
// emptying the cart after the order has been successful
fromPath.removeValue();
```

Figure 33 Creating an order history

The final step in this process is taking the employee to the home activity and letting them know that the order was successful with a toast message.



```
// informing the employee the order was a success
Toast.makeText(activity,  text: "Order completed!", Toast.LENGTH_SHORT).show();
startActivity(new Intent(getApplicationContext(), MainActivity.class));
```

Figure 34 Informing the employee the order was successful and starting the home activity

2.4 Graphical User Interface (GUI)

Login Screen:

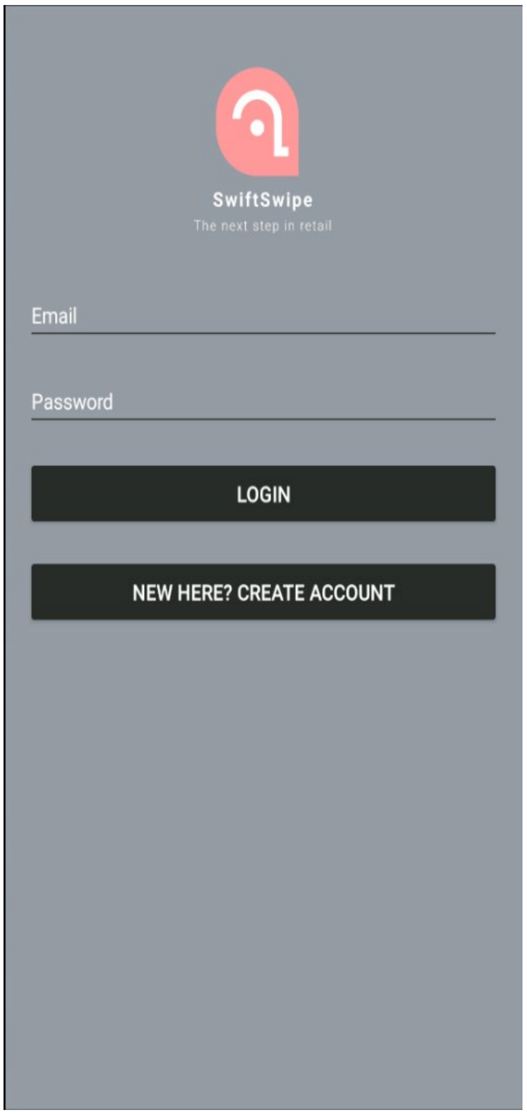


Figure 35 Login activity with no information

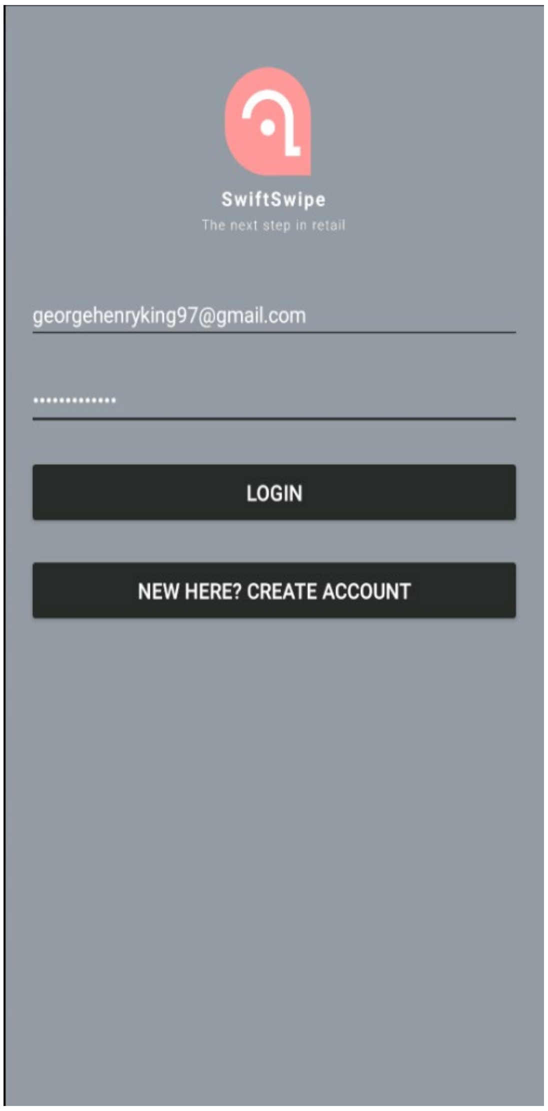


Figure 36 login activity with information

On the login screen there is a lot of functionality happening, but it is also simplistic in its nature. In order for a user to log in they are required to enter their email and the password associated with their account. There is some validation on these text fields such as requiring the password to be a certain length and checking if the fields have text in them before allowing a submission to happen. The information entered here is checked against the connected google firebase database. The other functionality on this page is the option to create an account if a person does not have one yet.

Register screen:

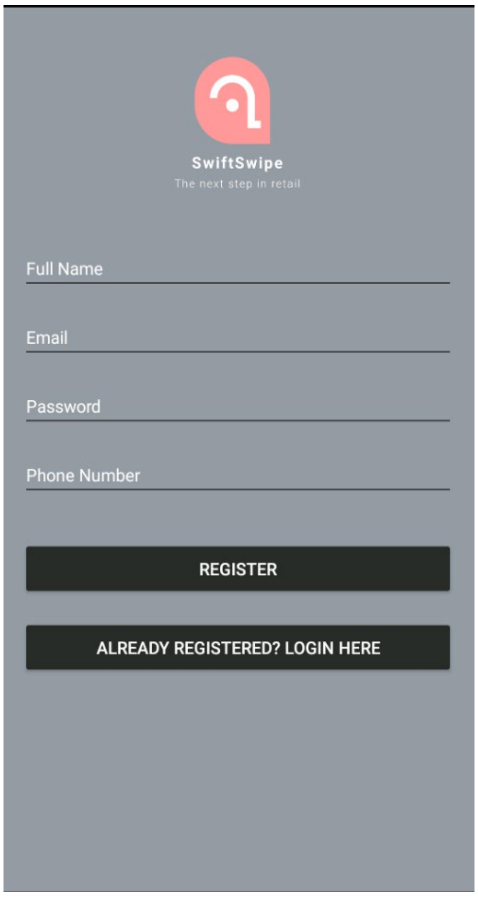
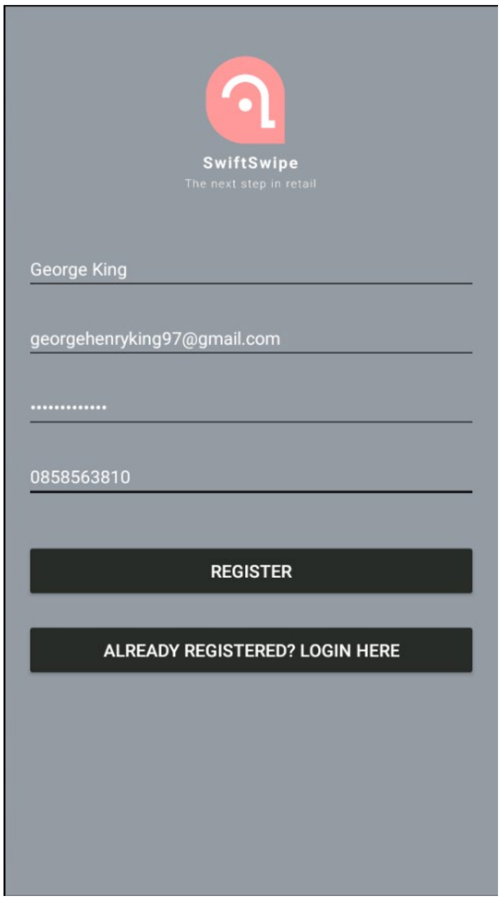


Figure 37 Register activity with no information



George King

georgehenryking97@gmail.com

•••••••••••••

0858563810

Figure 38 Register activity with information

On the register screen we allow a user to create an account to access the rest of the application. From the image you can see in order to create an account a user has to provide their full name, email, password, and phone number. The same validation rules are applied to these fields that are applied to the fields in the login page. Once the user has filled out the fields to the satisfaction of the system and they click register the account is created in the connected google firebase database and will then bring the user to the login activity. From there they are now able to log in to the application. The final functionality on this page is the already registered button. This is there in the instance a user clicks into this activity by mistake and would like to go back to the login activity which is where the user is brought if they click on the text.

Home activity:

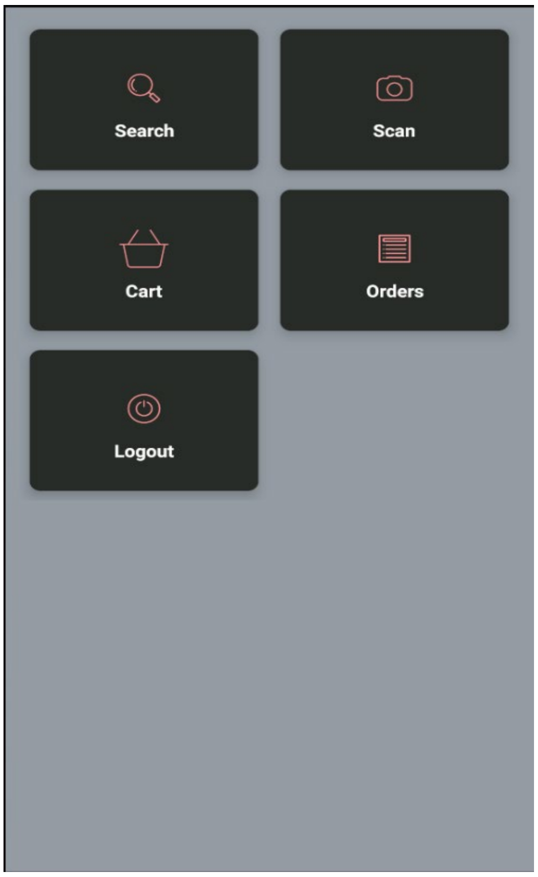
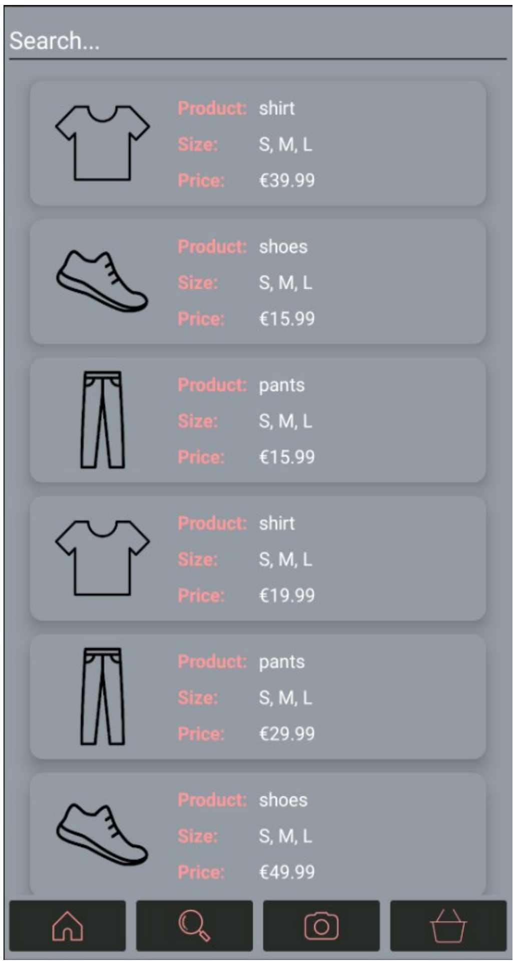


Figure 39 Home activity

On the home activity there is several activities that an end user can interact with. The first being the search activity. Clicking this button will take the end user to an activity where they can see all products of the store and give them the ability to search for a given product. The next activity is scan which allows a user to scan a barcode which will bring them to a product view of the barcode they just scanned. The third activity is cart which is where an end user can see all the products they have in their cart and checkout. The fourth is the orders activity where an end user can see all past orders they have made. Finally, an end user can also log out from the home screen which will then bring them back to the login page.

Search activity:


Search...

Product: shirt
Size: S, M, L
Price: €39.99

Product: shoes
Size: S, M, L
Price: €15.99

Product: pants
Size: S, M, L
Price: €15.99

Product: shirt
Size: S, M, L
Price: €19.99

Product: pants
Size: S, M, L
Price: €29.99

Product: shoes
Size: S, M, L
Price: €49.99

Figure 40 Search activity with no search made

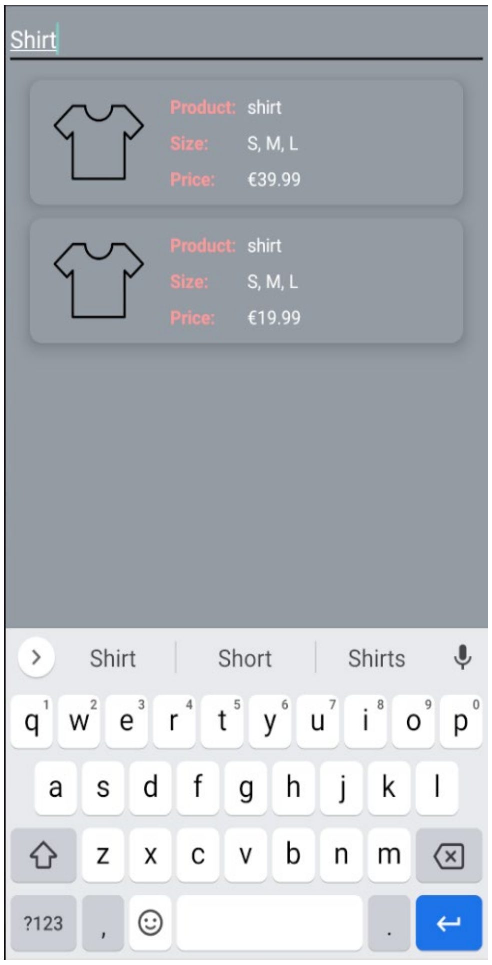


Figure 41 Search activity with search made

For the search view there is two main pieces of functionality available to the end user. The first being the search bar. With this search bar an end user can refine the products in the recycler view to whatever they specify. You can see in the image on the right how I have typed in shirts and all relevant results are shown. The second functionality is the ability to click into a product and be brought to a new activity where an end user can add that item to their cart. Also, you now see the introduction of the four buttons that are at the bottom of most pages on the application baring the register, login, home, and checkout activities. These buttons bring you to four of the main pages of the application. These being the home screen, the search activity, the barcode scanner, and the cart.

Product activity:

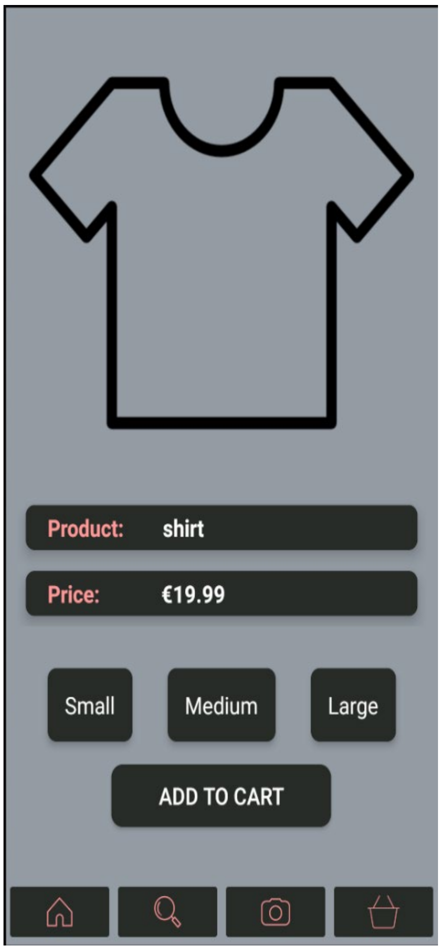


Figure 42 Product activity with no size selected

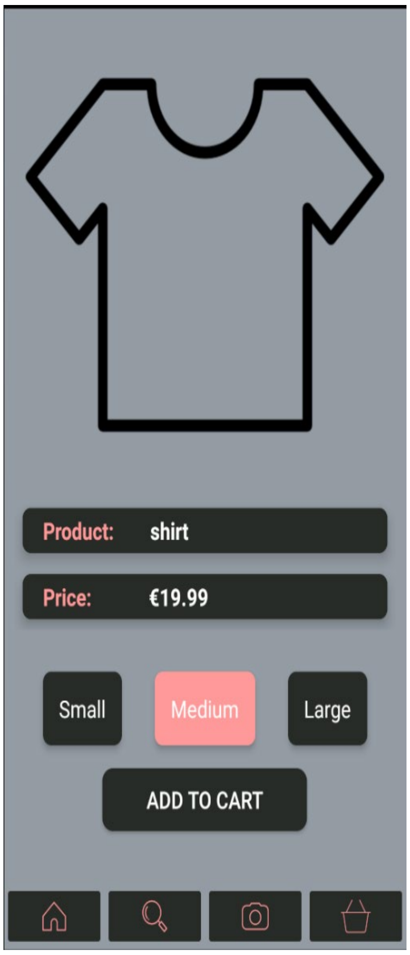


Figure 43 Product activity with size selected

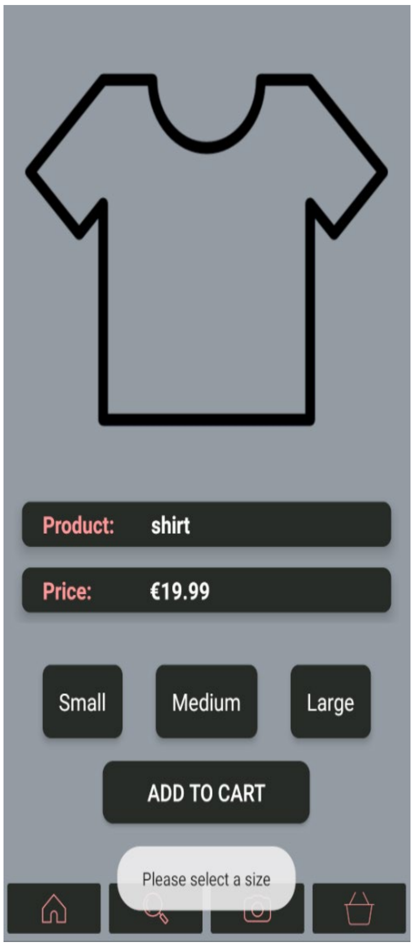


Figure 44 Product activity when an employee tries to add without selecting a size

For the product view it's kept mostly simple in functionality with the only two options being available to the user are size choice and adding to the cart. The end user can see the product name and the cost of the product. They can also see all sizes are available for the product and have their choice of which size they want to pick. If an end user tries to add to the cart without selecting a size they are prompted with a message asking them to select a size before trying to add to cart.

Scanner Activity:

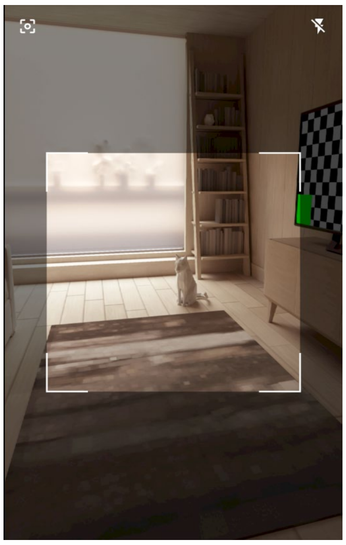


Figure 45 Template of what the barcode scanner view would look like

For the barcode scanner activity, the functionality is very simple. It shows the current view the camera on the device has and allows the scanning of barcodes. The idea behind this functionality is that when the user scans the barcode, they will be presented with the individual activity view that a user would get if they clicked into a result from the search function. In order to use this activity, the user must allow the application to have access to the device's camera.

Cart:

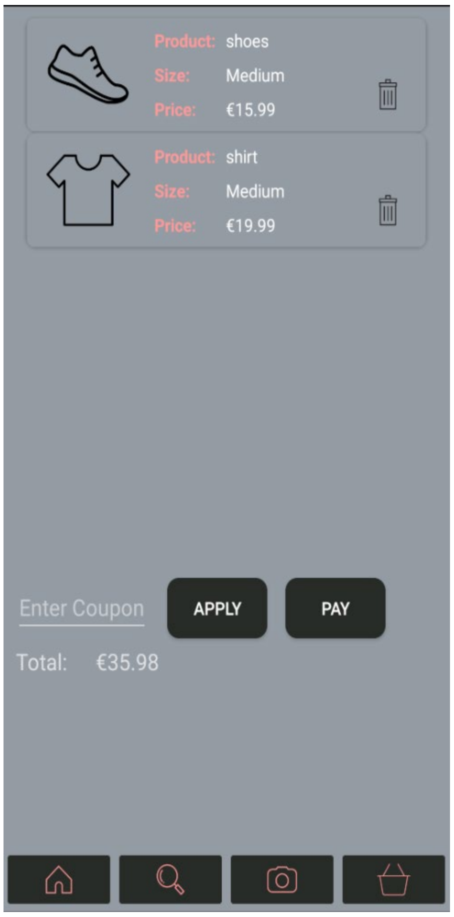


Figure 46 Cart activity with products

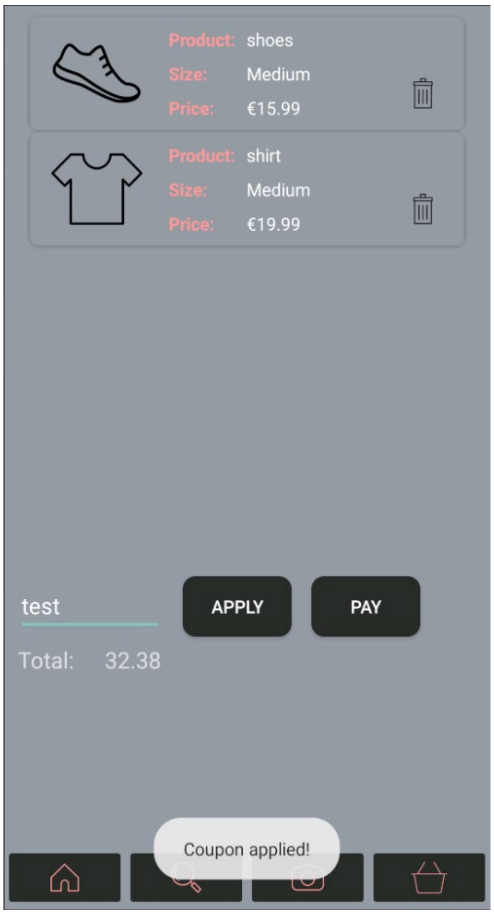


Figure 47 Cart activity with a coupon being applied

For the cart activity there is three choices the end user has to them. The first being the ability to delete items from their cart. All the user has to do is click the small bin icon on the right side of an individual view which will then remove it from the cart. The second choice is the ability to apply a coupon to the cart. In the second image you can see how this functionality works. The end user inputs the coupon and hits apply and if the coupon exists the system lets the user know with a message. The same happens if the coupon does not exist and the message is changed to coupon does not exist. The final choice the user has is the option to proceed to checkout with the pay button. The end user also has a recycler view at the top of the activity that shows all products currently in their cart.

Checkout:

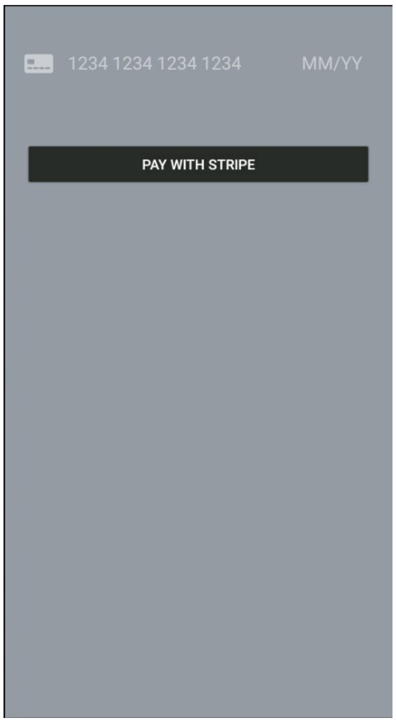


Figure 48 Checkout activity with no card information

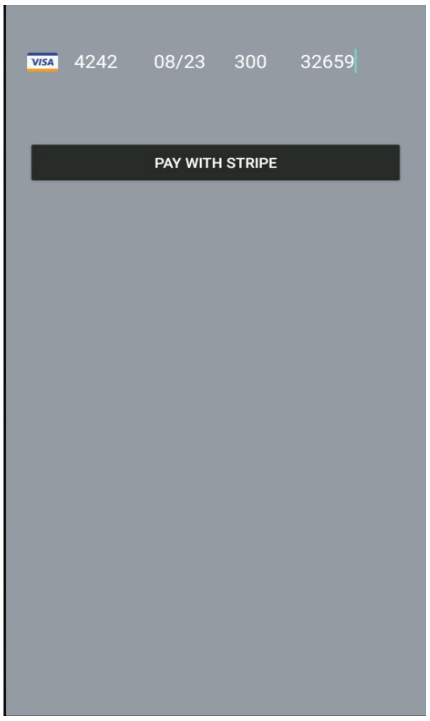


Figure 49 Checkout activity with card information

For the checkout activity there is only one option for the end user and that is to put in their credit card details and to press pay. If they don't wish to go through with the order they can press back and it will bring them back to their cart activity.

Orders:

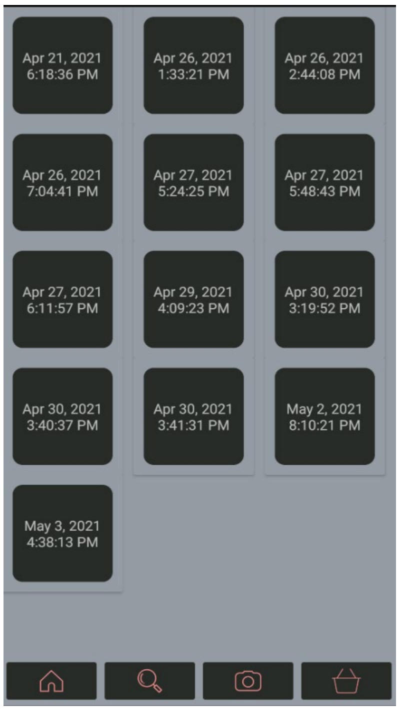
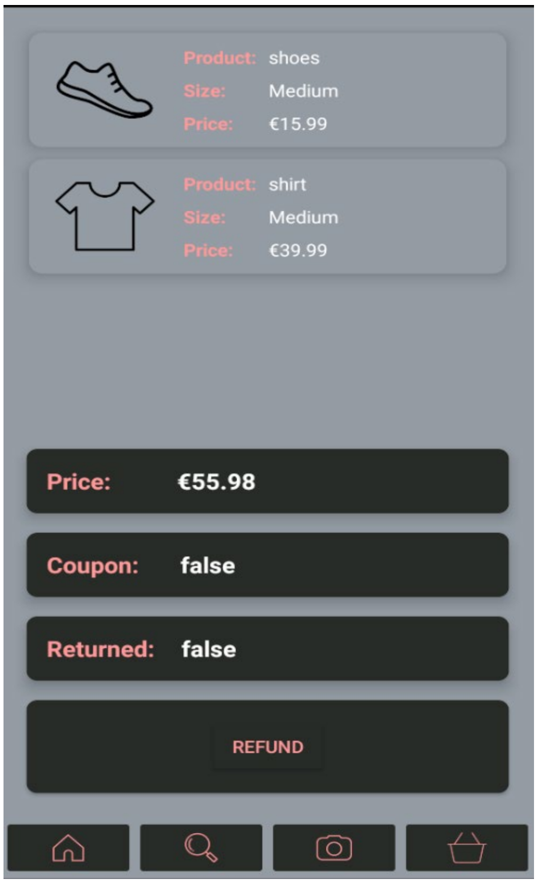


Figure 50 Order history list activity

For the orders page the end user can see all their past orders that they have made. They are able to click into each order individual if they want which will bring them to a detailed view of the order. Each order is defined by the date and time it was made on.

Order view:



Product: shoes
Size: Medium
Price: €15.99

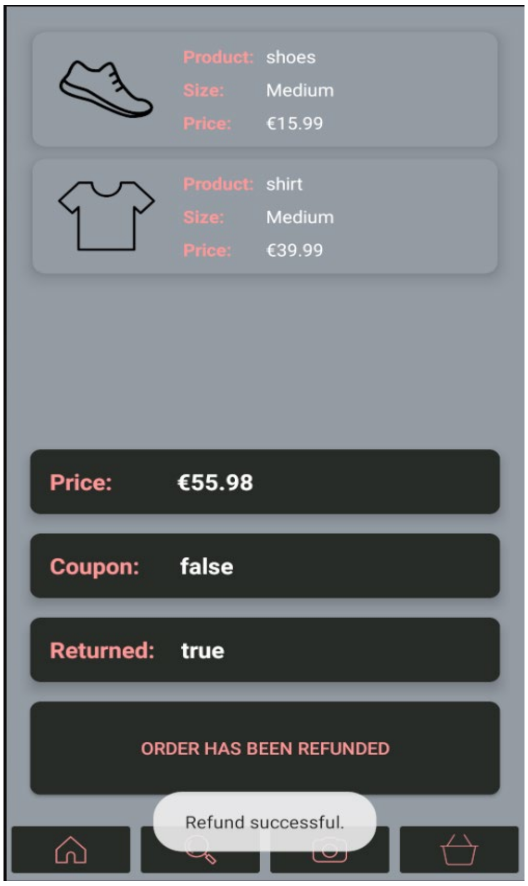
Product: shirt
Size: Medium
Price: €39.99

Price: €55.98

Coupon: false

Returned: false

Figure 51 Order history with no refund


Product: shoes
Size: Medium
Price: €15.99

Product: shirt
Size: Medium
Price: €39.99

Price: €55.98

Coupon: false

Returned: true

ORDER HAS BEEN REFUNDED

Refund successful.

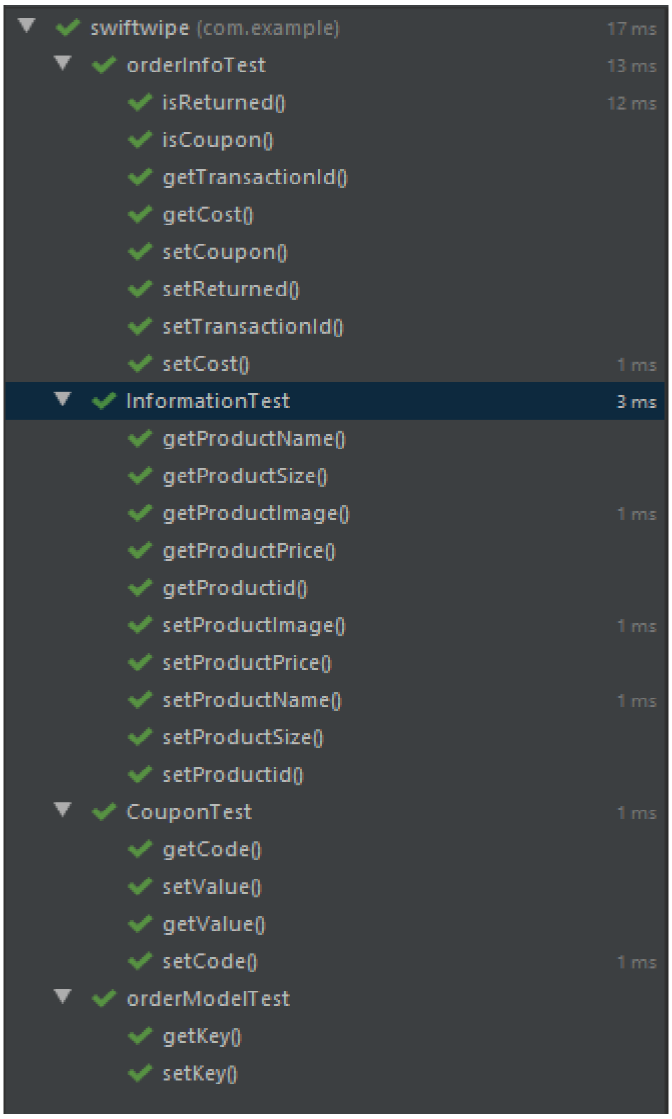
Figure 52 Order history with refund

From this view the end user has a lot of information provided to them. They can see each individual product that was bought and the price per item as well as the size. From there they can then see the total price of the order if a coupon was used or if the order has been returned. The end user also has the option to refund the entire order if they wish to do so. If the button is clicked the end user is presented with a message letting them know the refund was successful and the button is disabled so the user cannot try click the button again prompting another refund request.

2.5 Testing

For the purposes of testing my project I used a combination of Junit 5 and espresso. These testing languages allowed me to develop all the local unit tests that I needed for my system. However due to a large amount of my system functionality relying on interaction between firebase and it, Junit only made up a small portion of my overall testing coverage. It was mostly used for ensuring model classes were working as intended.

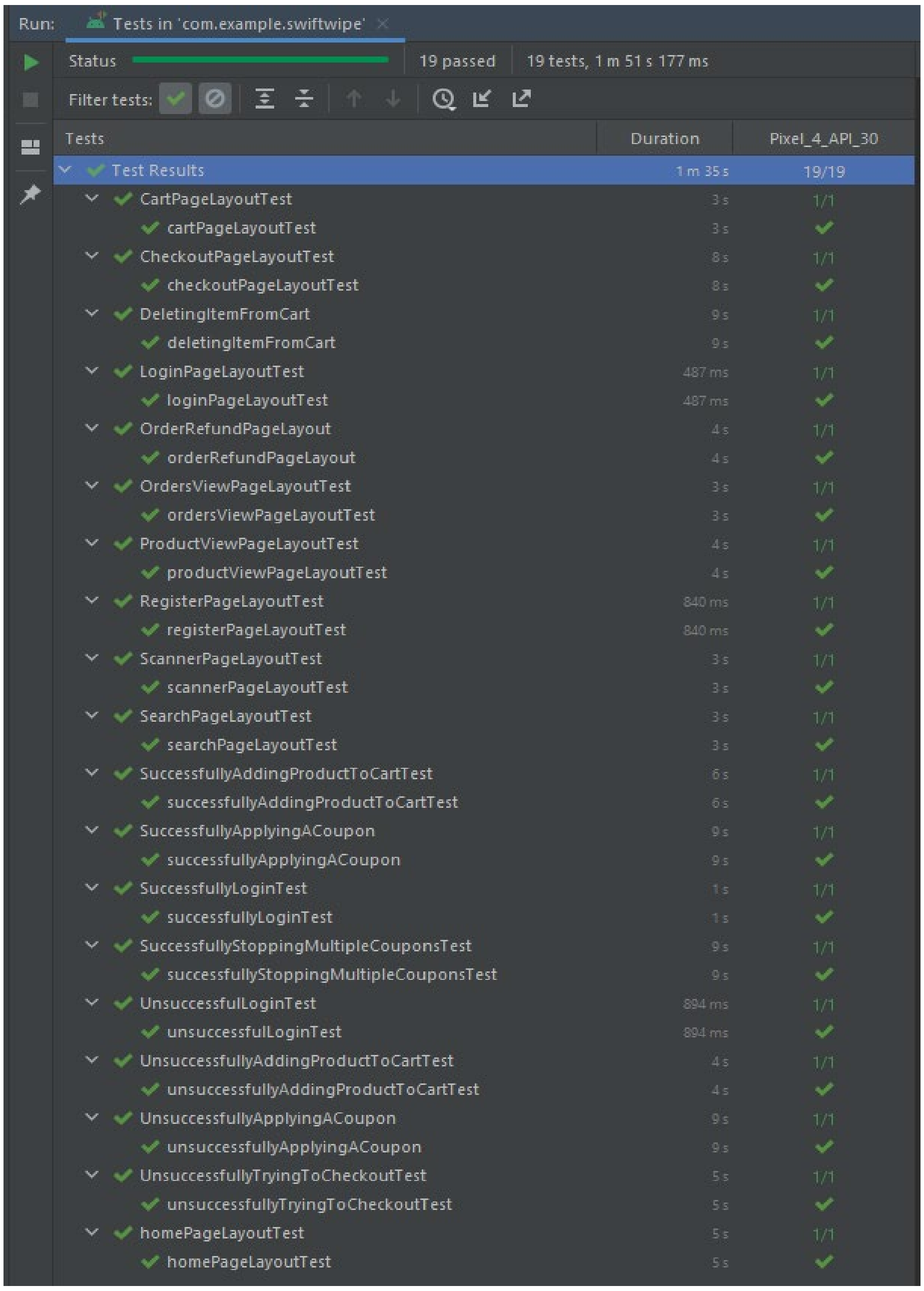
For my system there was a total of four model classes. The first being a coupon model, which contained a string for the coupon code and an int for the percentage to discount by. The second being an information model, which contained all the information required for a product in the store such as name, price, size, image, and ID. The third being an order info model, which contained information relating to an order such as transaction ID, cost, if it has been returned or a coupon has been used. The final model class is an order model object which consisted of one variable that being the key. For each of these model classes a Junit 5 test was made and all methods in the classes were tested.



```
▼ ✓ swiftwipe (com.example)          17 ms
  ▼ ✓ orderInfoTest                  13 ms
      ✓ isReturned()                 12 ms
      ✓ isCoupon()
      ✓ getTransactionId()
      ✓ getCost()
      ✓ setCoupon()
      ✓ setReturned()
      ✓ setTransactionId()
      ✓ setCost()                     1 ms
  ▼ ✓ InformationTest                 3 ms
      ✓ getProductName()
      ✓ getProductSize()
      ✓ getProductImage()             1 ms
      ✓ getProductPrice()
      ✓ getProductid()
      ✓ setProductImage()             1 ms
      ✓ setProductPrice()
      ✓ setProductName()              1 ms
      ✓ setProductSize()
      ✓ setProductid()
  ▼ ✓ CouponTest                      1 ms
      ✓ getCode()
      ✓ setValue()
      ✓ getValue()
      ✓ setCode()                     1 ms
  ▼ ✓ orderModelTest
      ✓ getKey()
      ✓ setKey()
```

Figure 53 Unit tests for model classes

In order to test my firebase functionality, I used the espresso testing language. Espresso is a step through testing language which allows creating complex tests simple. The testing language listens in the background as you move through the system following a use case and creates tests to ensure the same functionality happens again. I also used espresso to ensure that the layout of each activity was as intended and that all elements were on the page. In total I wrote a total of 19 tests with espresso. With nine of those tests being related to functionality and the other ten being layout related. While espresso is designed for layout testing with some not so conventional ideas it can be used to ensure that certain functions are performing as intended. For instance, in order to test my login functionality is working I tell the test to type in the correct information and click login. I then tell it to look for an element that is only on the home screen page of the system. If the system runs this test and cannot find that element, I know there is something wrong with my login activity.


```
Run:   Tests in 'com.example.swiftwipe'
Status                19 passed   19 tests, 1 m 51 s 177 ms
Filter tests:
Tests                                          Duration   Pixel_4_API_30
Test Results                                     1 m 35 s   19/19
  CartPageLayoutTest                                  3 s   1/1
    cartPageLayoutTest                                3 s   ✓
  CheckoutPageLayoutTest                              8 s   1/1
    checkoutPageLayoutTest                            8 s   ✓
  DeletingItemFromCart                                9 s   1/1
    deletingItemFromCart                              9 s   ✓
  LoginPageLayoutTest                              487 ms   1/1
    loginPageLayoutTest                            487 ms   ✓
  OrderRefundPageLayout                               4 s   1/1
    orderRefundPageLayout                             4 s   ✓
  OrdersViewPageLayoutTest                            3 s   1/1
    ordersViewPageLayoutTest                          3 s   ✓
  ProductViewPageLayoutTest                           4 s   1/1
    productViewPageLayoutTest                         4 s   ✓
  RegisterPageLayoutTest                           840 ms   1/1
    registerPageLayoutTest                         840 ms   ✓
  ScannerPageLayoutTest                               3 s   1/1
    scannerPageLayoutTest                             3 s   ✓
  SearchPageLayoutTest                                3 s   1/1
    searchPageLayoutTest                              3 s   ✓
  SuccessfullyAddingProductToCartTest                 6 s   1/1
    successfullyAddingProductToCartTest               6 s   ✓
  SuccessfullyApplyingACoupon                         9 s   1/1
    successfullyApplyingACoupon                       9 s   ✓
  SuccessfullyLoginTest                               1 s   1/1
    successfullyLoginTest                             1 s   ✓
  SuccessfullyStoppingMultipleCouponsTest             9 s   1/1
    successfullyStoppingMultipleCouponsTest           9 s   ✓
  UnsuccessfulLoginTest                            894 ms   1/1
    unsuccessfulLoginTest                          894 ms   ✓
  UnsuccessfullyAddingProductToCartTest               4 s   1/1
    unsuccessfullyAddingProductToCartTest             4 s   ✓
  UnsuccessfullyApplyingACoupon                       9 s   1/1
    unsuccessfullyApplyingACoupon                     9 s   ✓
  UnsuccessfullyTryingToCheckoutTest                  5 s   1/1
    unsuccessfullyTryingToCheckoutTest                5 s   ✓
  homePageLayoutTest                                  5 s   1/1
    homePageLayoutTest                                5 s   ✓
```
Figure 54 Espresso tests passing

2.6 Evaluation

How was the system evaluated and what are the results? This may consist of usage data. It may also include performance evaluations, scalability, correctness, etc. depending on the focus of the project. Quantitative results may be reported in tables or figures.

One of the main ways I am going to evaluate my application is by comparing the goals I set out to achieve and compare it against with the final product. The first of these goals is the functional requirements of the system. To recap them there was six in total these being searching the stockroom, creating a sale, and cancelling a sale, applying a discount, customer paying and refunding a customer. Five out of the six requirements were fully achieved with searching the stockroom being the one that did not get completed. It fell short in regard to being able to see the current amount of stock an item has and in what sizes. With this in mind I had a success rate of 83% when it came to completing my requirements if we do not consider searching the stockroom complete at all. If we consider that requirement half-finished this number moves up to 92%. Overall, I am happy with the level of success I achieved with my first goal.

My next goal was to have a UX/GUI design that was modern and usable. When looking at usability for a UI there is a lot of elements that need to be considered. The first step in this is 5Es of usability. This is a process where after establishing a problem you take these five steps to ensure the problem is solved with the users experience in mind. The five steps are:

- Effective
    - This step is to watch a user perform a task and see how many mistakes they make before being able to complete the task
- Efficient
    - This step is to give a user a task and to time them to see how long it takes to complete the task
- Engaging
    - This step is about drawing a user into their task and ensuring no screens are confusing.
- Error Tolerant
    - This step involves creating a test where an error is likely to happen and see how well a user can recover from it.
- Easy to Learn
    - This step involves giving the user as little information as possible and seeing how well they perform using your application.

While I worked on my systems UI, I constantly kept these ideas in mind and would ask myself these questions again and again each time I developed a new activity or layout to use. The next step I took in evaluating my UI was via Nielsen's heuristics. These are a set of ten rules of thumb you should follow when developing an application to ensure its usability regardless of a user's capability. These are as follows:

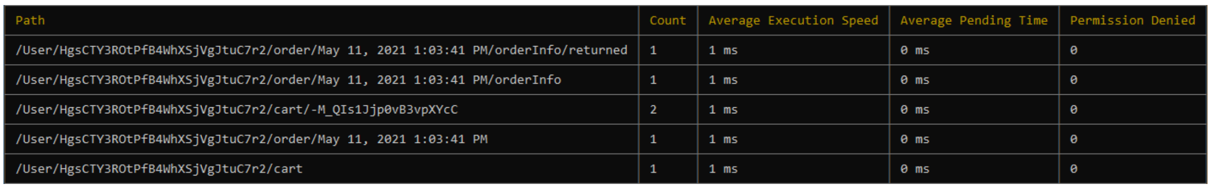
- Visibility of system status
    - I achieve this by informing the user with a loading icon when signing into their account from the login page. I also provide them with messages letting them know if functions have been completed or failed to complete.
- Match between system and the real world
    - Language across the system is kept simple and in the realm of what an individual in a retail store would expect to encounter.
- User control and freedom
    - Whenever a user wishes to stop a process all they have to do is click the back button and they are taken out of the process.
- Consistency and standards
    - Throughout the system word repetition is kept to the same meaning and icons never differ in where they bring you.
- Error prevention
    - Users are not allowed to add items to the cart without selecting a size first. They are unable to sign up until all the fields have been filled out correctly.
- Recognition rather than recall
    - When a user adds an item to their cart, they can see it straight away with all the same information they saw on the previous page
- Flexibility and efficiency of use
    - A user does not have to go to the home screen in order to go to the four main functionalities of the system, they can get access to them from any page via the four buttons on the bottom of every page.
- Aesthetic and minimalist design
    - Provided information is kept to a minimum and colour scheme is consistent throughout using only a three-tone colour scheme.
- Help users recognise, diagnose, and recover from errors
    - When a user does not select a size before they try to add to the cart, they are informed they need to. When a user does not properly enter card details, they are told. When a user tries to log in with the wrong information they are told.

- Help and documentation
    - Using hints in text fields where users will enter information such as when signing up, logging in, searching for information, or typing in card details.

Another aspect to evaluating my project is my test coverage and ensuring functionality works as need features are added. As it stands right now, I only have unit test coverage on my model classes using Junit 5 where all tests are passing. Firebase offers a way to test functionality that your system interacts with, but I was unable to figure this out before the deadline.

When looking at my database and how I could evaluate it I decided upon using firebases CLI software. This software has a built-in feature that will monitor your database as it is used. While not a perfect evaluation of the overall database it is a start. Once you stop the monitoring it will provide several pieces of information that are nice to know such as time for requests, request sizes and upload speeds.

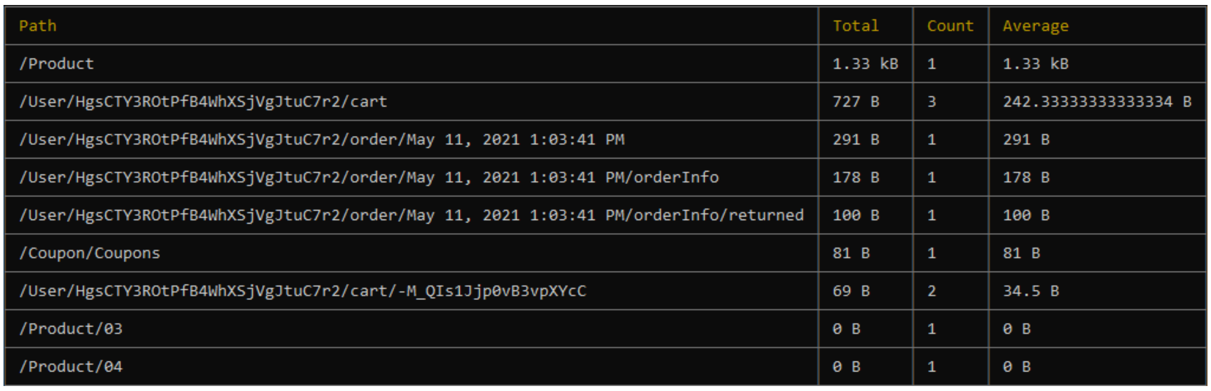
At various points in the system functionality, I have to write to my database. The image below shows a typical use case of my system where the user views a product, adds it to their cart and views the order after completion. You can see the speed of these executions is excellent at only 1 millisecond per write.



Path	Count	Average Execution Speed	Average Pending Time	Permission Denied
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/order/May 11, 2021 1:03:41 PM/orderInfo/returned	1	1 ms	0 ms	0
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/order/May 11, 2021 1:03:41 PM/orderInfo	1	1 ms	0 ms	0
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/cart/-M_QIs1Jjp0vB3vpXYcC	2	1 ms	0 ms	0
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/order/May 11, 2021 1:03:41 PM	1	1 ms	0 ms	0
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/cart	1	1 ms	0 ms	0

Figure 55 Write speeds to various locations on the firebase database

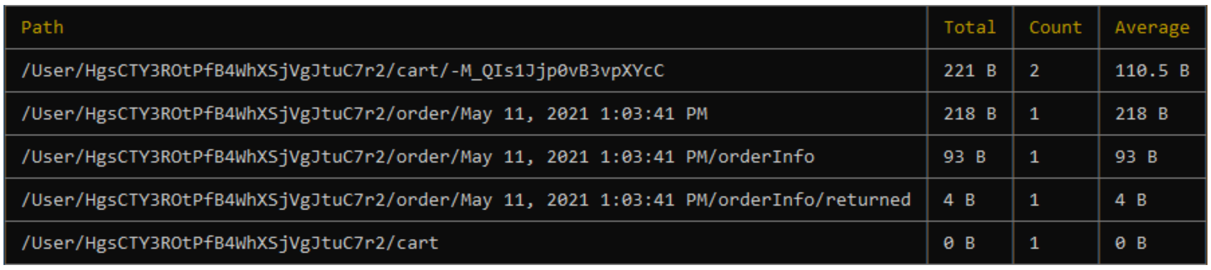
I also have to download from the database in order to populate certain views on my system. It is important to keep these sizes as small as possible as data can be costly on performance. With this in mind I kept my download sizes to a minimum, with the largest download only being 1.33kB.



Path	Total	Count	Average
/Product	1.33 kB	1	1.33 kB
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/cart	727 B	3	242.33333333333334 B
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/order/May 11, 2021 1:03:41 PM	291 B	1	291 B
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/order/May 11, 2021 1:03:41 PM/orderInfo	178 B	1	178 B
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/order/May 11, 2021 1:03:41 PM/orderInfo/returned	100 B	1	100 B
/Coupon/Coupons	81 B	1	81 B
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/cart/-M_QIs1Jjp0vB3vpXYcC	69 B	2	34.5 B
/Product/03	0 B	1	0 B
/Product/04	0 B	1	0 B

Figure 56 Download sizes from firebase database

The final element of my database evaluation is my upload speeds. In order for user changes to persist if the system is closed is to upload these changes to the database. These changes mostly revolve around the system state and items the user has in their cart. The largest upload being only 221 bytes, this being a user adding an item to their cart.



Path	Total	Count	Average
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/cart/-M_QIs1Jjp0vB3vpXYcC	221 B	2	110.5 B
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/order/May 11, 2021 1:03:41 PM	218 B	1	218 B
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/order/May 11, 2021 1:03:41 PM/orderInfo	93 B	1	93 B
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/order/May 11, 2021 1:03:41 PM/orderInfo/returned	4 B	1	4 B
/User/HgsCTY3ROtPfB4WhXSjVgJtuC7r2/cart	0 B	1	0 B

Figure 57 Upload sizes from firebase database

The final way I will be evaluating the project is by its performance on a device. Given that my application is developed for android devices I will not know exactly what sort of hardware the system is run on. With this in mind I kept its hardware requirements at a minimum. The two elements I will be looking at is CPU and memory usage through a simple use case of the system. Similar to the use case used for evaluating the database above for the sake of consistency.

With this in mind we can see from the below image that CPU performance never goes above 45% and this is only the case when starting the application. However, CPU usage fluctuates a lot and often drops to a zero percent state during normal use of the application.

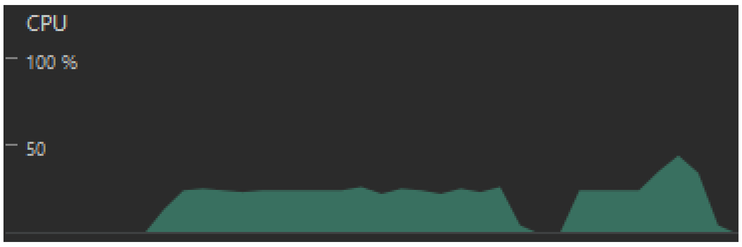


Figure 58 CPU usage when starting the application

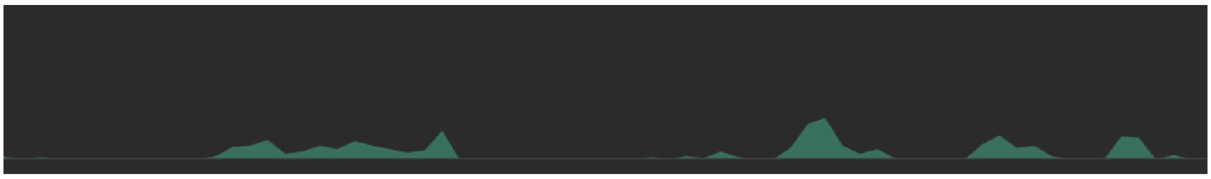


Figure 59 CPU usage during normal use of the application

In regard to memory usage the maximum amount of memory used is 144mbs when starting the application. For the entirety of the program the memory usage fluctuates between 100mbs and 144mbs. Compared to the CPU usage the memory is more consistent with its usage never.

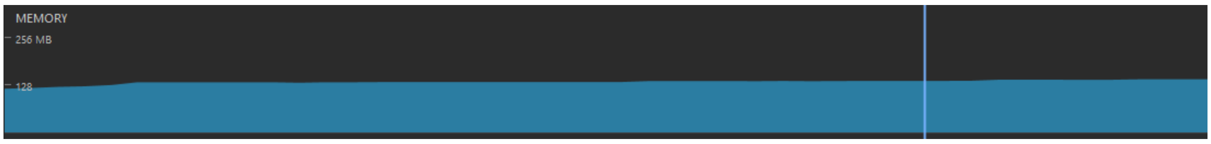


Figure 60 Memory usage during normal use of the application

Overall, the project has its flaws, but it also does excellent in other areas and gives the opportunity to improve it going forward.

3 Conclusions

In conclusion there is a lot I achieved with this project and I am happy with the result of it. Countless hours of research and effort went into the current state of the system and I can be proud of the submission in its final state. One of the main advantages I had when developing this system was the countless amount of information readily available online that allowed me to create the functionality I required. With most of it coming at no cost to me which was very helpful and allowed me to elevate my project to a higher standard that would not have been achievable if it were not for this case.

For the final submission of my project is covers most of the goals I set out to achieve. The first being allowing a user to search for products that are in the store. Initially I had the idea that they could see how many of that product was in stock but overtime the scope of the goal changed and in its final state it only allows a user to see products in the store. This could be seen as a limitation of the project but there is the option on the admin side that if a product is out of stock, it can be removed from view easily enough. While it is not very user friendly the functionality is still technically there. The next goal of my project was to allow a user to add an item to the cart which I believe I achieved fully. When a barcode is scanned the ID is taken from it and the individual product view is show to the user which they can then add to the cart. The next goal I had in mind was a cart system where a user could add and remove products, apply coupons and checking out which I also achieved. There were some bugs with the cart that I would have liked to fix but due to time limitations was unable to do so and these will be discussed in the further development & research stage. Finally, the last goal of my project was to facilitate card payments on my application via NFC technology. I would say I achieved half of this goal with my system being able to facilitate card payments. However, it is not with NFC and a customer would have to manually enter their card details to facilitate the payment. This is not ideal for a real-world application and large security concerns come into play. Stripe, the system I am using for my payments, does have the option to implement services like apple pay or google pay which would allow me to accept card payments but again due to time limitations I was unable to do so.

As far as limitations of my project there is a view that come to mind with most being real world applications. The one technical limitation I found that was not caused by time was my search functionality. Firebase only allows you to query with exact values and not dynamically. This was disappointing as I had envisioned a dynamic search functionality but by the stage, I made it to this feature I had already invested a lot to the technical stack I had chosen. Other than that, most of the limitations I found came when I questioned how my application would actually tie into a real-world scenario. I noticed as time went on, I made large assumptions about the implementation of the project in the real world . Such as a store switching over to my service completely rather than just integrating to the system they currently have. After having realised this, I noticed it would be a large obstacle to tackle from the outside rather than developing the system in house. Having no knowledge of how they format their system flow or what else I would have to interact with in order for my application to be considered viable for a store. Not to say that it would be impossible to do I quickly realised how challenging it would be and decided to try stick to my goals and create a system mock-up of sorts that I could show to potential interested parties and then from that point onward we could customise it to their needs and what they wanted.

4 Further Development or Research

Working on this project over the past year has been such an amazing experience both in terms of technical growth and how I approach my work. I have learnt so much throughout the course of my workings and if given the opportunity to continue working on this project is a lot, I would do to improve the overall quality and potential prospect of the project.

The first step I would take is reaching out to retailers and asking them about their current system. Finding out how they set it up and how their system flows so I could get a better understanding of how I could integrate my system into theirs. I think this is a key takeaway because my application can offer as much convenience to their customer and added versatility to their staff in their duties but if they cannot integrate it easily it is unlikely it would get far in the process of being adapted. Hopefully from this research I could find some common elements between different retailers and their system flows so that adapting it to multiple stores with unique set ups would not be overly complex. Such as having to use different technologies depending on the store.

The next step would be creating features I want to but did not have time to, there is a lot of these I considered and have kept the ones I think would stay inside the scope of what this project is aiming to achieve while also increasing its usability to its users. The first function I would include is an employee profile section. In this section employees would be able to do several things. Including requesting time off, seeing their current work schedule, advertising a shift for swapping, and seeing companywide announcements for franchised stores. The idea behind this function is it provides value to the employees of the store and makes it more versatile to their needs as well. Not requiring a separate area for employees to find that information.

The next function I would add is a store section. In this section employees could see information relating to their own store such as opening hours for the holidays, store wide announcements versus companywide announcements or a checklist of actions that have to be completed on a daily basis. The idea behind this section is things are always changing with people calling out or leaving to pursue other things. A store could put up posts about job opportunities or advertise extra shifts. As well with the checklist with opening and closing if you are unsure of what needs doing you would be able to check there.

The next function I would include is a stock checking feature. With it being an original goal of the project, I think it would be important to include both as a function and as a completeness aspect of the project. Also, it heavily impacts the reduction of employee count aspect of my project. If the store still needs someone to check stock before creating the sale, I have no reduced employees where possible.

The final large function I would add on is a dual log in system. This would be a large task as the functionality for admins would include several things such as creating coupons that employees can use, advertising shifts, setting schedules, removing, or adding products, updating stocks etc. I do think it would be a great feature that has a lot of value for potential buyers. Making interacting with the system overall much easier and pleasant to deal with.

The next few things I would add are smaller in nature and I will provide them in a bullet point list below.

- Categories on the search menu
- Dynamic search bars
- On the orders list refine by date or cost
- Google pay/Apple pay via Stripe
- Barcode scanner would pass in the ID and size of the product
- Refund individual items of an order rather than requiring the entire order to be refunded
- Remove all items from a cart at once if a customer changes their mind

- Restructure order history to be independent of the employee who made it
- Update the coupon error messages to be less generic and provide more accurate information

There are also the known bugs that are currently in the system. These are listed below but currently only affect two use cases of the system.

- When deleting an item from the cart the total price is not updated
- On certain activities when the devices back, button is pressed it brings you to a random activity
- Layouts do not fit on every device as they use hard coded values rather than weighting

Overall, there is a lot left I want to do with this project and plenty of opportunity to live up to the goals I set out to achieve. Going forward I see companies starting to adopt this outlook and refitting how they use their employees and make their stores more dynamic. Rather than just having set roles in a store and a set place to pay. In theory the idea is sound but implementing it successfully from an outside perspective would certainly come with its challenges.

5 References

GitHub. 2021. *yuriy-budiyev/code-scanner*. [online] Available at: <https://github.com/yuriy-budiyev/code-scanner> [Accessed 15 May 2021].

Stripe.com. 2021. *Custom payment flow*. [online] Available at: <https://stripe.com/docs/payments/integration-builder> [Accessed 15 May 2021].

Youtube.com. 2021. *Before you continue to YouTube*. [online] Available at: <https://www.youtube.com/playlist?list=PLVW1e1FvhW67aJ2alJePjwYtlivd8klv9> [Accessed 15 May 2021].

Youtube.com. 2021. *Before you continue to YouTube*. [online] Available at: <https://www.youtube.com/watch?v=sZ8D1-hNeWo> [Accessed 15 May 2021].

6 Appendices

6.1 Project Plan

National College of Ireland

Project Proposal

SwiftSwipe

08/10/20

BSc (Honours) in Computing

Software Development

2020/2021

George King

18106188

X18106188@student.ncirl.ie

Contents

1.0 Objectives

When thinking about the objectives of my project there are several that come to mind. Each of these objectives is paramount to the success of my application. They have been carefully considered and critically analysed to ensure the necessity of them. It was important to me that the objectives I have for my project added value to the customers experience and were not just needless features. This was also done to avoid scope creep.

The objective first being making the customer experience as seamless as possible and having to only deal with one employee. I plan on doing this by facilitating all current functionality provided by a store of employees to being provided by just one employee and my application. In turn this lead into my next objective of reducing the number of employees needed in the store. In order to make my application business viable I need it to reduce costs of the businesses using it. I hope to achieve this by reducing the number of employees needed as well as data gathering for marketing purposes. I will reduce the number of employees needed by giving every employee in the store the capabilities required for the store to work. Thus, removing specific roles such as a checkout employee and allowing employees to always be working. The next objective I hope to achieve is facilitating payments anywhere in the store to deal with queues. This objective came to mind when thinking about Christmas shopping and how you can spend longer queueing than shopping. I aim to achieve this objective by retrofitting the devices currently used in stores with the capabilities of NFC payments. The final objective that I have is data gathering for the businesses. I hope to achieve this by various methods but currently the only one that I have thought of is email gathering for providing receipts. Gathering data like this has a cost associated with it and with my application providing a new way to gather it I have made my application more feasible for businesses to consider.

2.0 Background

My idea come from a hackathon I did during my internship last summer. For the hackathon we were tasked with coming up with new utilisations of NFC and how it can be used to allow small businesses to take card payments. We were broken up into teams of four and given two weeks for preparation work to flesh out our idea and then a day to create a presentation and a video talking about your idea. The presentation consisted of a mock-up UI and talking about the various features of our idea. At the end of the presentation, we got asked questions by the three judges. Thankfully, the questions we got asked we had already considered and come up with answers for. Overall, it was a great experience and our idea ended up winning the hackathon.

One concern we had for the application at the time was the current spending limit on tap payments. Currently set at 50€ we had to consider how we would deal with larger expenses and security concerns if we were to hypothetically raise the spending limit. I raised this concern with the committee hosting the hackathon and made the argument that we are slowly heading towards a world where most transactions are being done via card and so the raising of the tap amount was inevitable. They agreed and decided to not take that issue into concern when considering ideas. However, in terms of my project and real-world viability it is not as easy to deal with. I do think that concern is slightly out of my control though and I intend to focus on what I can control with my project. While this issue will make my application less viable for stores where purchase amounts on average are greater than 50€, going forward the limit will be raised and as a result my application will then become viable.

The reason I want to continue to explore this idea and pick it for my final year project is because I believe it has a strong chance of being a useful product with business viability. With the hackathon I did not get the opportunity to fully develop the idea and create a software side to it. With the idea being my final year project, I will get to have that opportunity and see it all the way through.

When looking at other applications that replicate the same functionality as my own there is not an officially released one. However, there is a case study currently going on in south America which has similar aims and objectives to that of my own project. As the case study is currently on going there has yet to be any officially released results. However, researchers involved in the study have commented that the results look promising. It is possible to look at what is currently in use for store payments/checking stock. Traditionally or currently when wanting to pay in a store the customer has to make their way to a physical location in the store to do so. Similarly, when wanting to check stock in certain stores they have stock employees in the stock room of the store who can be reached by earpiece. Whereas in other stores an employee would have to go back there themselves to check.

3.0 Technical Approach

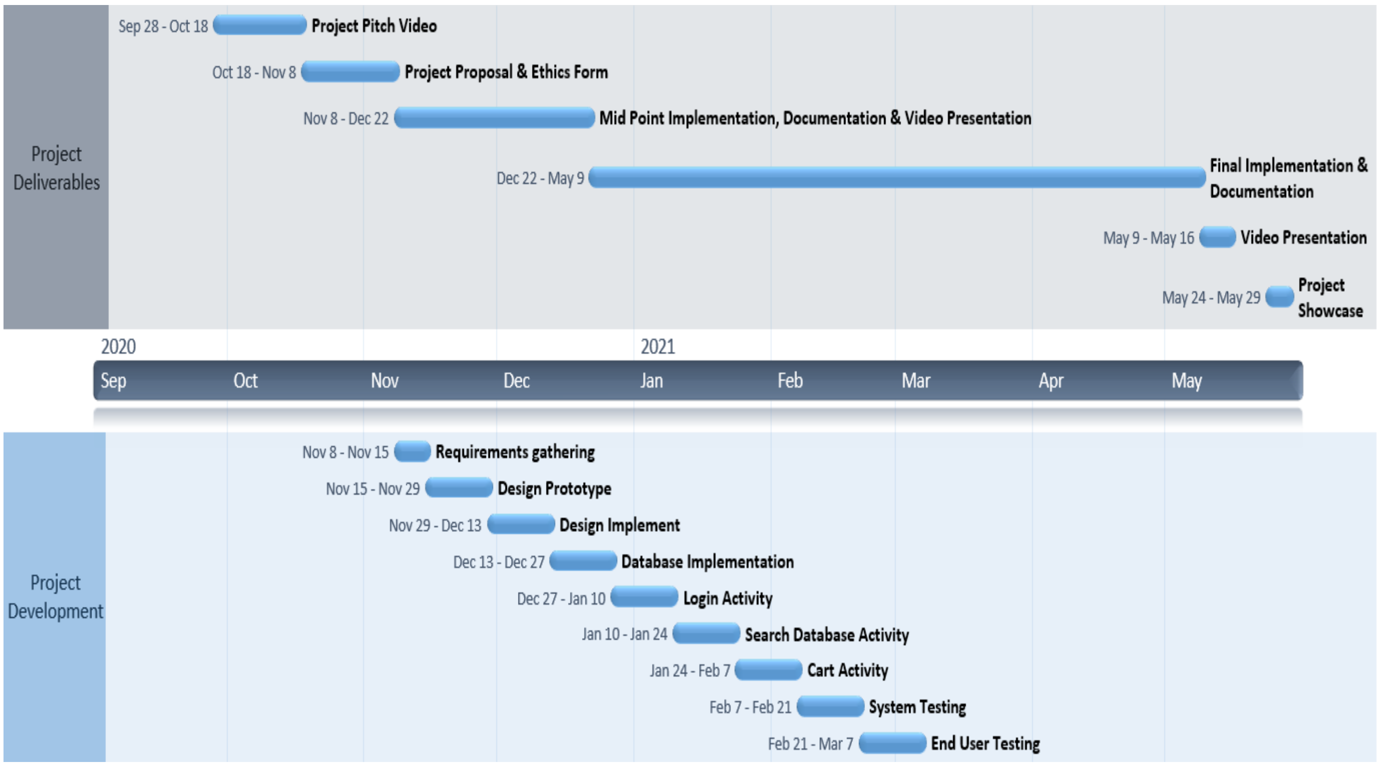
When considering the approach to take when developing out my application I only had two methods in mind, that being the waterfall method and the agile method. While waterfall is generally considered outdated and not practical in the modern world it does have it is advantageous for small projects. With my project being just me and no other participants involved the requirements and other deliverables are not going to change, so waterfall is a valid consideration for the approach to take. However agile can also take advantage of this fact and I will more than likely be using it in my career going forward. For this reason, I think the right choice is to use Agile to get more experience using it. It will benefit me moving forward and has the same advantages that waterfall does. While not having the drawbacks of waterfall if my supervisor advises me to try push the scope of my project, which is likely given our initial meeting.

With this decision in mind the first step is to research applications currently on the market to get an idea of what is there. From there I will be able to see which features are on offer and what is not on offer. From there I can move on to requirements gathering both functional and non-functional to know what I will need my application to have. With those captured I can start to mark out the timeline for the features and when I should have them done. Included in this will be reflective views at midpoints throughout the project as well as testing at the end of the project to ensure correct functionality throughout the system.

4.0 Special Resources Required

The main special resource I will require is an ability to mock NFC payments. I am currently unsure how to go about this but will do more research surrounding this topic when it comes closer to the time of testing. The other special resource I will require is an android device as my personal device is using iOS software. While I can run my unit tests and see how the design will look via the emulator built in android studio it is still important to be able to see how the application will look in person on a device or at a production level.

Project Plan



5.0 Technical Details

When deciding on which language I wanted to use I thought of it from the perspective of how I wanted to interface with the customer and due to my application requiring mobility I went with a mobile application. This led to the choice of iOS or android for which platform I wanted to develop on. Given my previous experience in Java and the wider usage of android devices I decided that android would be the best platform to use. With this in mind the final decision I had to make was Java or Kotlin since you can develop with either language for android applications. Like I said above having had a lot of previous experience in Java I felt Java was the best bet as I want to develop the best application that I can.

Another language I will need to use is one for testing. Anytime I have tested before it has been done manually but part of the project deliverables is using test units to verify the functionality of my project. Having considered this, I have done some small amounts of research on various testing languages and have decided on Junit.

When thinking of libraries that I will require for my project there are many within the scope. The first being the libraries required for NFC payments. As well as the libraries required for my firebase database.

6.0 Evaluation

When considering how I wanted to evaluate my system I decided that the best way would be to incorporate a testing language to mock any type of input my system would expect. In order to do this, I will be implementing the Junit language into my project. With this language I will be able to ensure that every aspect of my project is working as it should and I can ensure that when I submit my project for final review that it will be in working condition. This would include the log in system, the cart system, the payment system, and the database integration.

One issue I can see myself running into when evaluating my application is mocking the NFC payment aspect. I have yet to be able to find any information regarding this topic and it is an area of concern moving forward. I hope to be able to find a way to mock the payment by the time I reach that stage of my project.

The final evaluation that needs to be considered is the experience from the end users' perspective which would be an employee in a given store. To achieve this, I will require an ethics form which will allow me to get feedback about the system flow and design so that I can improve it.

6.1 Reflective Journals

6.1.2 October

My feelings for the first month of my software project have been mixed. The first part of this process was the video proposal. Overall, I found this assignment easy enough. Answering the questions proposed to us in the assignment specification allowed me to flesh out my idea and to consider parts of the idea that I had not before. From there the wait to hear back about my idea getting approved or not was a stressful one due to the timeframe with the next assignment on the 8th of November. Thankfully, it all worked out in the end and my idea did get approved and I feel like I still have enough time to complete that assignment before the due date.

One concern I do have is that when I was working on a mobile project for another class, I found that my mobile application skills were lacking in certain areas that I would require for my project. To combat this, I have started doing some example projects to get better at my mobile development. I am hoping this will put me in a better spot once I get to the development stage of my project.

6.1.3 November

For the month of November, I was happy with the progress I made on my project. I had talked with my supervisor about various concerns that I had, and she was very helpful in tackling them. The main concern I had was time management with a heavy workload besides the final year project. Other than my concerns I have been able to make progress in the way of setting up my login and connecting my database for my app. I have also started work on the barcode and going forward I need to link that with my database too. Overall moving forward with the project, I am concerned about the midpoint submission and how much work I have let to do until I am ready for it. I am hoping to find time with finishing other assignments sooner than their deadline but that is currently not working out too well.

6.1.4 December

For the month of December there was a lot that happened with my final year progress. The midpoint submission was due on the 22nd and there was a lot I wanted to get done for that submission. Thankfully, I had about a five-day period where I could exclusively work on the codebase and report which allowed me to make a lot of progress. Overall, I was not super happy with the state the application was in for the submission, but I felt it satisfied the requirements of the submission. I was also apprehensive about doing the video recording as having done some for other modules earlier in the term they had proved to be difficult and requiring many takes. Thankfully for this submission though I found my previous experience doing the videos to be beneficial and was able to complete the recording after three takes.

We were also informed that the next semester workload with be lighter due to only having three modules on top of the final year project. Feeling good about that prospect with the amount of work I have left to do. I am also keen to take advantage of the holiday break to make as much progress as possible once all my other assignments are in from the 6th of January onwards.

6.1.5 January

6.1.6 February

For the month of February in regard to my final year project I made some good progress. I got my database working for many sections of the project as well as started to look at my card reader. Found an interesting library that may facilitate the functionality that I want. Need to discuss further with my supervisor how to move ahead in regard to this.

Overall, very happy with the current state of my project and the current speed at which its going, due to less classes overall the project has become far less stressful and can see the final product.

6.1.7 March

6.1.8 April

6.2 Other materials used