**Deliverable 4: System Prototyping and Testing**

**Food Wastage Application: The Sustainable Spoonful**

**By**

**Lea Thumbiran, Lucinda Zachos, Melany Opperman, Santana Bradbury**

A Mini-Dissertation Submitted as a Partial Requirement for the Bachelor Science in Information Technology: Mobile Application and Web Services

In the Faculty of Information Technology, Eduvos

**Supervisor:**

Mr. Ian Masaga

Date: 15 June 2023

1. **Student Details**

|  |  |  |  |
| --- | --- | --- | --- |
| Student Details | Student Number | Telephone Number | Email Address |
| Opperman, Melany | 6P6NPJX46 | 074 709 6778 | 6p6npjx46@vossie.net |
| Thumbiran, Lea | LMTPQFTH6 | 079 898 2004 | lmtpqfth6@vossie.net |
| Zachos, Lucinda | Y34VR5C17 | 071 878 3138 | Y34vr5c17@vossie.net |
| Bradbury, Santana | CNZJB3199 | 071 864 4170 | cnzjb3199@vossie.net |

1. **Supervisor Details**

|  |  |  |
| --- | --- | --- |
| Supervisor Name | Faculty | Email Address |
| Masaga, Ian | IT | ian.masaga@eduvos.com |

**Table of Contents**

[Chapter 4: System Prototype Development and Testing 4](#_Toc138671499)

[4.1 Introduction 4](#_Toc138671500)

[4.2 Testing Plan 6](#_Toc138671501)

[4.3 Layouts Development 14](#_Toc138671502)

[4.3.1 User Interface – Main Screen 14](#_Toc138671503)

[4.3.1.1.1 Iteration 1 14](#_Toc138671504)

[4.3.1.1.2 Iteration 2 15](#_Toc138671505)

[4.3.2 User Interface – Registration 16](#_Toc138671506)

[4.3.2.1.1 Iteration 1 17](#_Toc138671507)

[4.3.2.1.2 Iteration 2 18](#_Toc138671508)

[4.3.3 User Interface – Login 21](#_Toc138671509)

[4.3.3.1.1 Iteration 1 21](#_Toc138671510)

[4.3.3.1.2 Iteration 2 22](#_Toc138671511)

[4.3.4 User Interface – Retail Partners 24](#_Toc138671512)

[4.3.4.1.1 Iteration 1 25](#_Toc138671513)

[4.3.4.1.2 Iteration 2 25](#_Toc138671514)

[4.3.5 User Interface – Discounts 28](#_Toc138671515)

[4.3.5.1.1 Iteration 1 29](#_Toc138671516)

[4.3.5.1.2 Iteration 2 29](#_Toc138671517)

[4.3.6 User Interface – Products 33](#_Toc138671518)

[4.3.6.1.1 Iteration 1 33](#_Toc138671519)

[4.3.6.1.2 Iteration 2 34](#_Toc138671520)

[4.3.7 User Interface – Account 35](#_Toc138671521)

[4.3.7.1.1 Iteration 1 36](#_Toc138671522)

[4.3.7.1.2 Iteration 2 36](#_Toc138671523)

[4.4 Business Logic Development 40](#_Toc138671524)

[4.4.1 Algorithm 1 – Login 40](#_Toc138671525)

[4.4.2 Algorithm 2 - Registration 42](#_Toc138671526)

[4.5 Data Access Development 45](#_Toc138671527)

[4.5.1 Database Implementation 45](#_Toc138671528)

[4.5.1.1 Database Creation Class – DatabaseHelper.java 45](#_Toc138671529)

[5.1.1 Data Access Adapters 52](#_Toc138671530)

[5.1.1.1 Accessor Methods – Account 52](#_Toc138671531)

[5.1.1.2 Accessor Methods – Products 53](#_Toc138671532)

[6.1 Conclusion 54](#_Toc138671533)

[List of Figures 56](#_Toc138671534)

[List of Tables 56](#_Toc138671535)

[Bibliography 57](#_Toc138671536)

# System Prototype Development and Testing

## Introduction

The main goal of The Sustainable Spoonful mobile application is to assist in mitigating food wastage amongst retailers and consumers.

During the system prototype and testing phase, the following factors had to be considered, these factors included creating a mobile application that was visually appealing, user-friendly and one that met the functional requirements that were initially stated at the start of the development process. After considering these factors, the prototype for how the system will function as well as the design diagrams for the system were developed.

The layouts development, business logic development, data access development and testing stages will be covered in this section:

The layouts development stage involves recreating the prototype from the previous iteration and building upon it. The prototype from the previous iteration includes the registration, login, store, discounted products, and account screens. During this stage the users’ experience must be taken into consideration (Soegaard, 2019). The mobile application should be easy to navigate through and be responsive on mobile devices (Soegaard, 2019). The design of the screens should remain consistent throughout the mobile application (Soegaard, 2019). Usability testing should be conducted to ensure that users can navigate and interact with the mobile application easily (Soegaard, 2019). This can ensure that users will have a positive user experience when using this mobile application (Soegaard, 2019).

The business logic development stage involves translating the business requirements into functional code (Fowler, 2017). During this stage it is crucial that the business requirements are clearly defined. Having clearly defined business requirements helps ensure that it can be easily translated into functional code that follows the industry’s best practices. In addition, it can ensure that the code is maintainable, readable, and scalable (Fowler, 2017). This can be achieved by adopting an agile approach and continuously improving the code throughout the development process (Fowler, 2017).

The data access development stage involves designing the data access layer that will interact with the mobile applications backend system and the SQLite database. Afterwards, testing will be conducted to ensure that the mobile application is reliable and of a high quality (Ng, 2021). During this stage, it is important that the integrity of the user and mobile application’s data is maintained and secured (Ng, 2021). It is also imperative that an efficient and scalable database is developed, this can assist in minimizing response times by optimizing data retrieval (Ng, 2021).

Lastly, during the testing process, it is important to establish a testing strategy, create test cases, ensure that all features and functionalities of the mobile application are tested and documented and report any issues that have been identified (Hamilton, 2019). By taking these steps, this can ensure that a stable and reliable mobile application will be developed (Hamilton, 2019).

During these various stages, different roles have been allocated to the development team, this ensures that tasks are divided and distributed evenly throughout the planning and development process. The roles are shown in Table 1 below:

|  |  |  |
| --- | --- | --- |
| **Roles and Responsibilities** | | |
| **Name** | **Role** | **Responsibilities** |
| Melany | * Frontend Developer * Proofreader | * Designing and developing the various screens of the mobile application. * Proofread all deliverable documents. |
| Lea | * Backend Developer * Proofreader | * Designing and developing the functional aspects of the mobile application. * Proofread all deliverable documents. |
| Lucinda | * Documentation * Testing | * Compile project documentation. * Testing the mobile application for errors and unexpected behaviour. |
| Santana | * Proofreader | * Proofread and researched. |

**Table 1: Roles and responsibilities**

The aim of this chapter is to apply our research towards building a mobile application that can make a difference in communities.

## Testing Plan

During this stage, various tests will be conducted to determine whether an average user can navigate through the mobile application with ease. The type of testing that will be conducted is usability testing. Various test cases have been developed for users to work through, once they have completed these test cases, they can complete a survey. The survey will provide the development team with insight into how users view the mobile application and how they wish that it can be improved.

**Objective:**

* Understanding how users will interact with the mobile application and making changes based on those results (Optimizely, 2023).
* Ensuring that the mobile application is effective, efficient, engaging, error tolerant and can be easily to learn how to use. This can include (Quesenbery, 2023):
  + **Effective**: How accurately were tasks completed and how often did they produce errors?
  + **Efficient**: Evaluate the time taken to complete tasks.
  + **Engaging**: Gauge the users’ feelings towards the mobile application with the use of surveys or interviews.
  + **Error tolerant**: Create test case scenarios that may have the potential to cause errors.
  + **Easy to learn**: Attempt to gather test users from different technical knowledge backgrounds to determine how user friendly the mobile application is.

**Scope:**

* This test will include all the screens in the mobile application, as well as the content of each screen and navigating between the different screens (Usability.gov, 2019).
* The screens in this mobile application are (Usability.gov, 2019):
  + Landing screen
  + Login screen
  + Registration screen
  + Home screen
  + Store screen
  + Discounted Products screen
  + Account screen

**Components:**

1. Set up a usability test consisting of various tasks that the user would need to complete. These tasks will include:
   * Creating an account.
   * Logging in to their account.
   * Browsing the available discounted products.
   * Viewing their account.
   * Logging out of the mobile application.
2. Set up a survey for users to complete after they have tested the mobile application (Optimizely, 2023).

**Equipment:**

* Testing will be conducted through the users’ mobile device.

**Test Cases** (Zinchenko, 2019):

**Test Case 1**: The user installs the mobile application and opens it for the first time. Upon opening it, the user would need to register for an account, after registering the user would use their credentials to login to the mobile application.

* In this scenario, the user would need to open the mobile application, register an account, login to the mobile application and then locate the discounted products screen.
* For test case 1 to be considered successful, the user needs to register an account, and then login to the mobile application.

**Test Case 2**: The user closes the mobile application after viewing the discounted products screen. If they re-open the mobile application, would the user remained signed in or would they need to re-enter their login details?

* In this scenario, the user would need to close the mobile application from the recently opened applications list on the device and then re-open The Sustainable food mobile application.
* For test case 2 to be considered successful, the user needs to open the mobile application without needing to log into the mobile application again.

**Test Case 3**: The user needs to navigate to the accounts screen using the menu. If they logout, will they be redirected to the landing screen?

* In this scenario, the user would need to navigate to the account screen, view their account information and then press the logout button.
* For test case 3 to be considered successful, the user would need to navigate to the account screen, logout and then be redirected to the landing screen of the mobile application.

**Test Case 4**: The user wishes to view a discounted item from Woolworths. Could they easily navigate to it?

* In this scenario, the user would need to navigate to the stores page, click on the Woolworths card and then select one of their discounted products.
* For test case 4 to be considered successful, the user would need to navigate to the discounted products section without requesting assistance and complete the action in a reasonable amount of time.

**Test Case 5**: The user wishes to navigate to the account screen and then navigate back to the discounted products page using the navigation menu. Would they be able to easily navigate through it?

* In this scenario, the user would need to navigate to the account screen and then navigate back to the discounted products screen.
* For test case 5 to be considered successful, the user would need to navigate to the account screen and then to the discounted products screen without requesting assistance and completing this action in a reasonable amount of time.

**User Survey Questions:**

A Google form has been created for users to complete once they have worked through the five test cases.

The link to the form is attached below:

<https://docs.google.com/forms/d/e/1FAIpQLSc3EnTE__Z76rWmMWVvez45fT4MDG6OvFQgRelu5jnNebCrZg/viewform>

**Test Report (Zinchenko, 2019):**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Usability Testing: Testing Schedule and Report – 18 June 2023** | | | | | | | |
| **User** | **User Description** | **Test Case 1** | **Test Case 2** | **Test Case 3** | **Test Case 4** | **Test Case 5** |
| 1 | Female, age 56 limited technical knowledge | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| 2 | Female, age 32, Firm grasp of technology | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| 3 | Male, age 33, Firm grasp of technology | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| 4 | Female, age 20, Firm grasp of technology | Close with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| 5 | Female, age 20, Firm grasp of technology | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |

**Table 2: Testing Schedule and Report**

**Analysis and recommendations:**

**Test Case 1:**

Analysis:

* 80% of the users were able to complete test case 1 without any issue.
* 20% of users were not able to complete test case 1 without any issues due to the mobile application lagging after registration.

Recommendations:

* Review the code related to the registration process and make changes accordingly.

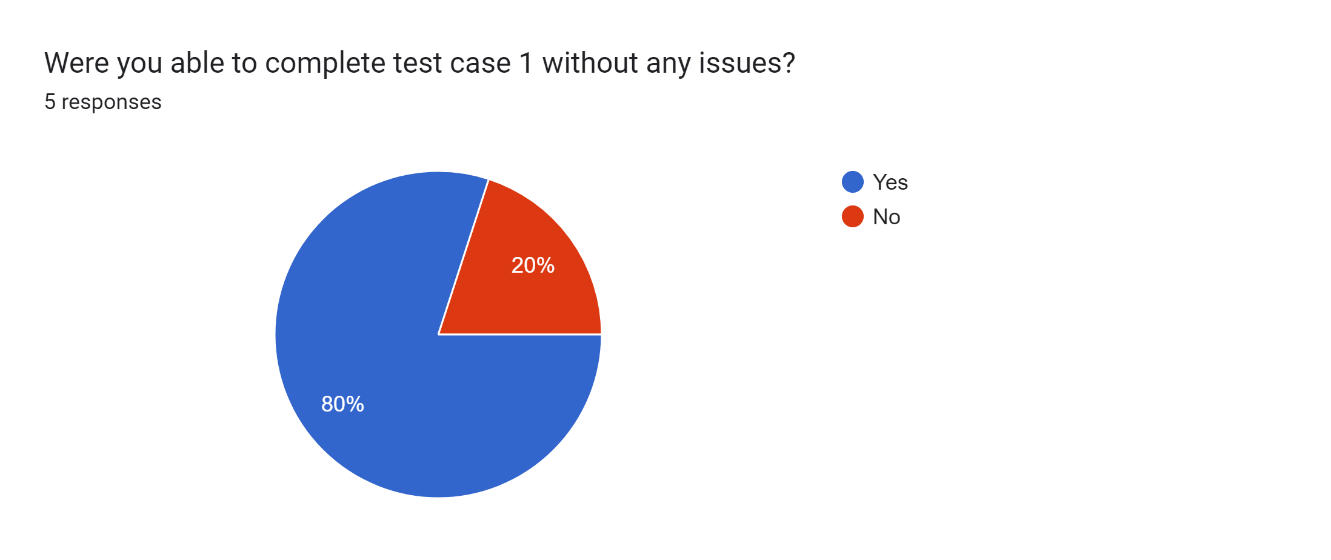


Figure 1 – Test Case 1 - Results Graph

**Test Case 2:**

Analysis:

* 100% of the users were able to complete test case 2 without any issue.

Recommendations:

* Keep building on the current iteration.

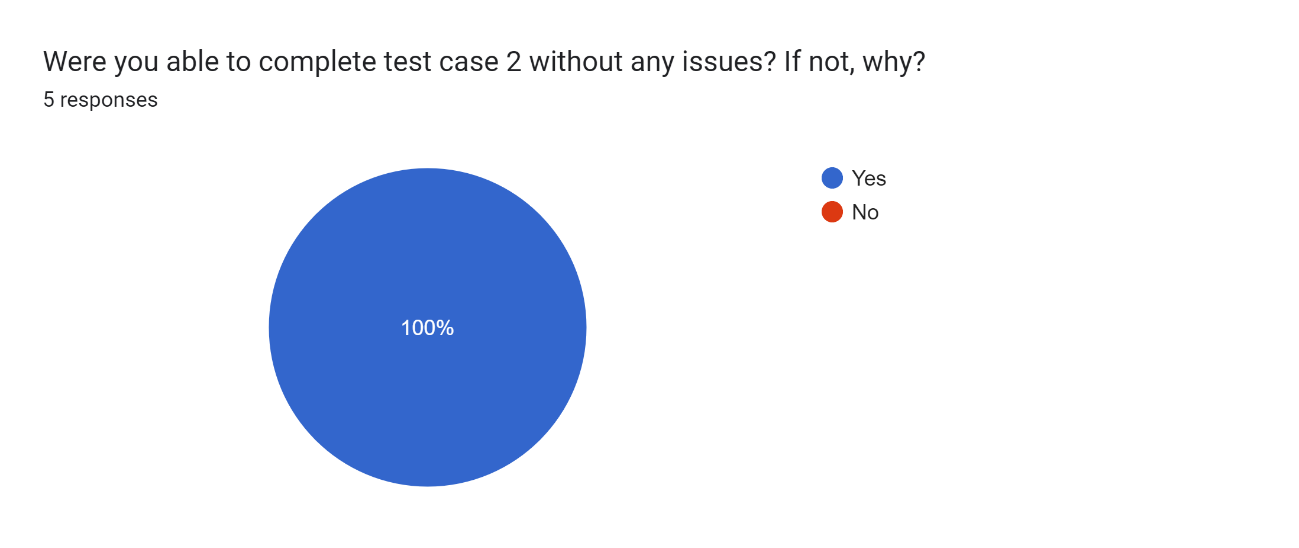
****

Figure 2 – Test Case 2 – Results Graph

**Test Case 3:**

Analysis:

* 100% of the users were able to complete test case 3 without any issue.

Recommendations:

* Keep building on the current iteration.

Forms response chart. Question title: Were you able to complete test case 3 without any issues? If not, why?
. Number of responses: 5 responses.

Figure 3 – Test Case 3 – Result Graph

**Test Case 4:**

Analysis:

* 100% of the users were able to complete test case 3 without any issue.

Recommendations:

* Keep building on the current iteration.

Forms response chart. Question title: Were you able to complete test case 4 without any issues? If not, why?
. Number of responses: 5 responses.

Figure 4 – Test Case 4 – Results Graph

**Test Case 5:**

Analysis:

* 100% of the users were able to complete test case 2 without any issue.

Recommendations:

* Keep building on the current iteration.

Forms response chart. Question title: Were you able to complete test case 5 without any issues? If not, why?
. Number of responses: 5 responses.

Figure 5 – Test Case – Results Graph

**Additional Responses:**

Analysis:

* 80% of the users stated that everything on the mobile application was readable.
* 20% of the users stated that the colours used made it difficult to read the test on screen.

Recommendations:

* Work on adjusting the colours to make the mobile application more readable.
* Consider implementing a dark mode for users who find the current theme too bright or difficult to read.

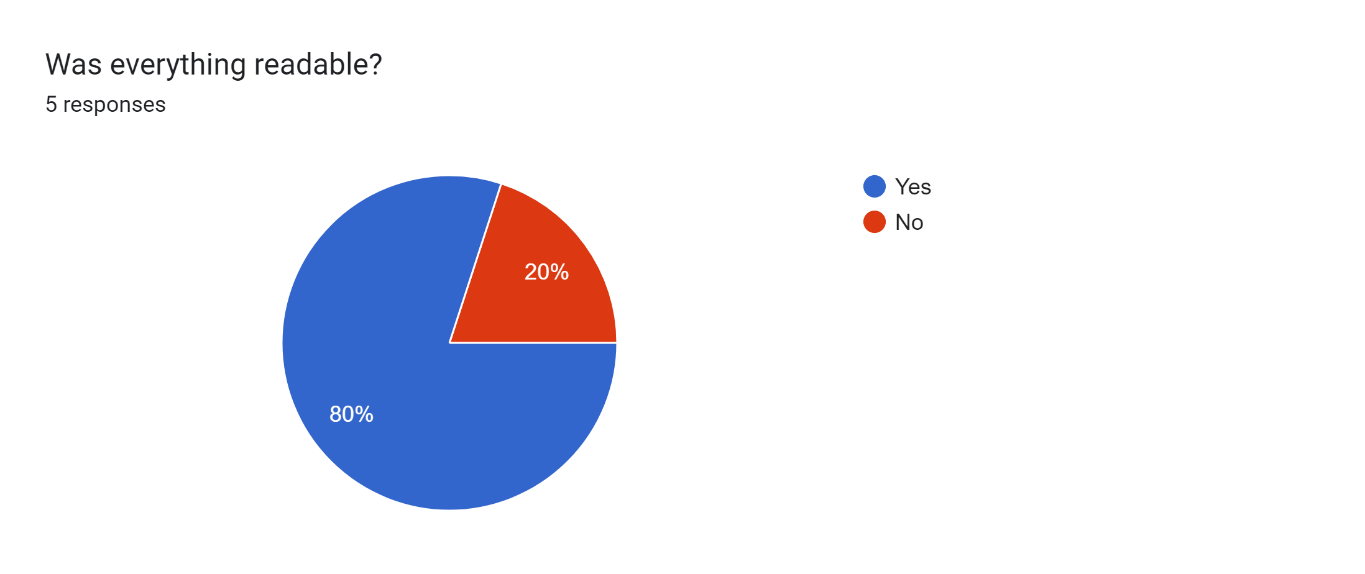


Figure 6 – Was everything readable - Result Graph

Analysis:

* 100% of the users stated the mobile application was responsive.

Recommendations:

* Keep building on current iteration.

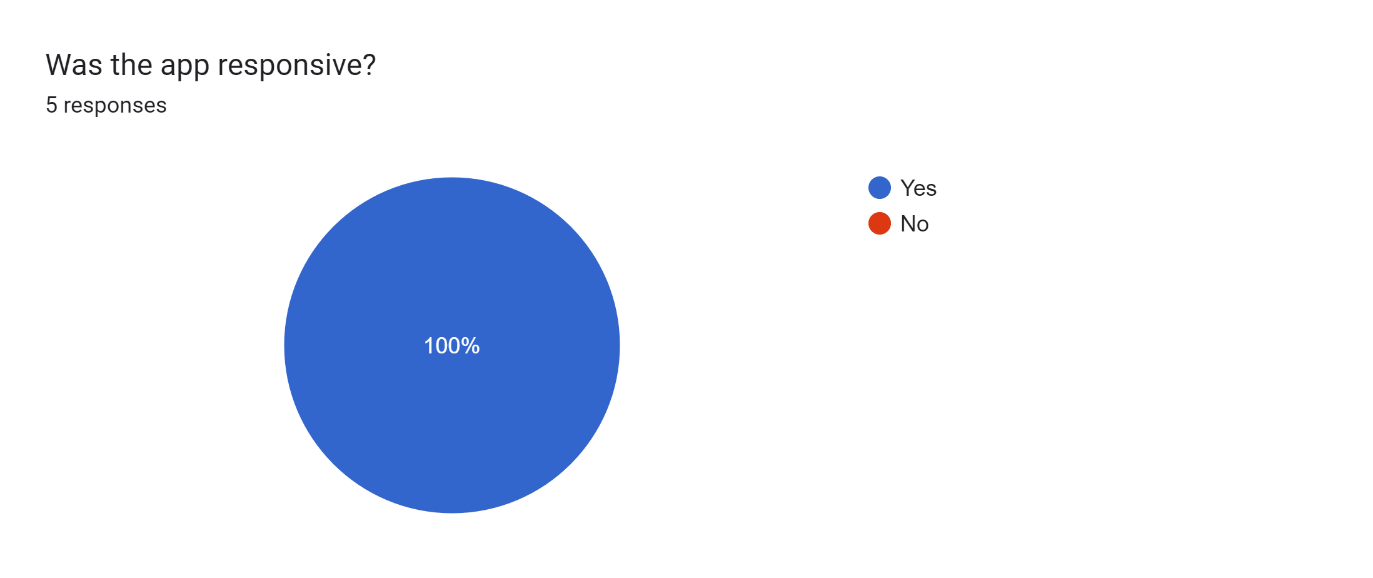


Figure 7 –Was the application responsive - Result Graph

Analysis:

* 100% of the users found the mobile application was easy to navigate.

Recommendations:

* Keep building on current iteration.

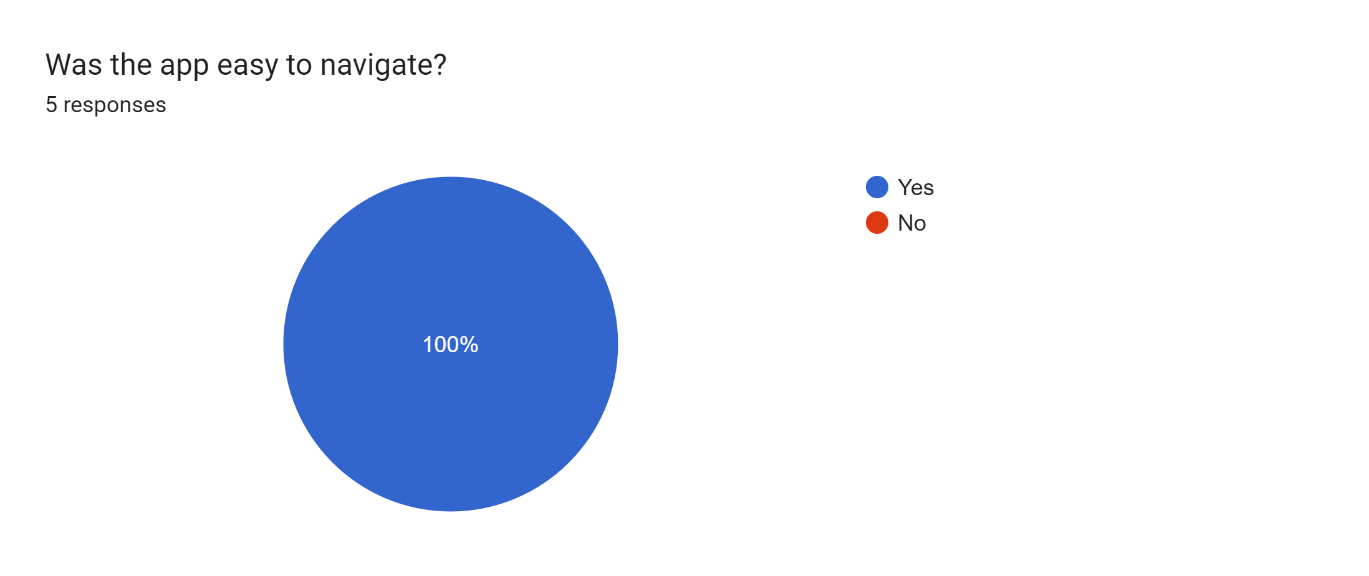


Figure 8 – Was the application easy to navigate – Result Graph

Overall, most users responded positively to using the mobile application and filling out the survey form. However, some users requested that they wish the mobile application had the capability to change themes, colours and enable them to edit their account details.

## Layouts Development

### User Interface – Main Screen

This is the first screen that users will view when opening the mobile application for the first time. Users are presented with a simple, user-friendly interface that has a logo and two buttons where they can either register for an account or login with an existing account.

If the user has already registered an account and logged in, upon opening the mobile application again, the user will automatically be redirected to the home page. This removes the need for repetitive login procedures.

By offering both the login and registration options on this initial screen, both types of users (such as new users an existing users) will be catered to. This will ensure that there is a personalized experience for all.

The first iteration of the initial screen as well as the current iteration are shown below:

##### Iteration 1

A screenshot of a blue screen

Description automatically generated with low confidence

*Figure 1 - Landing screen when the user opens the mobile application for the first time.*

##### Iteration 2

A screen shot of a cell phone

Description automatically generated with medium confidence

*Figure 2 - Landing screen when the user opens the mobile application for the first time. The mobile application is running on the Emulator Pixel 6 Pro API 30.*

**Code Snippet:**

package com.example.sustainablespoonfulapp;  
  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.database.CursorWindow;  
import android.os.Bundle;  
import android.preference.PreferenceManager;  
import android.view.View;  
import android.widget.Button;  
  
import java.lang.reflect.Field;  
  
public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*); //Create the main page when opening the application:  
  
 try {  
 Field field = CursorWindow.class.getDeclaredField("sCursorWindowSize");  
 field.setAccessible(true);  
 field.set(null, 100 \* 1024 \* 1024); //increases size to 100mb  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 //Checking if the customer is logged in:  
 SharedPreferences sharedPreferences = PreferenceManager.*getDefaultSharedPreferences*(getApplicationContext());  
 boolean isLoggedIn = sharedPreferences.getBoolean("isLoggedIn",false); //Set is logged in to false:  
  
 //If the customer is logged in, redirect to the home page:  
 if(isLoggedIn) {  
 Intent intent = new Intent(MainActivity.this, LandingActivity.class);  
 startActivity(intent);  
 finish();  
 }  
  
 /\*Navigating to the Registration Page After Clicking the Register Button\*/  
 Button registerButton=findViewById(R.id.*home\_register\_button*);  
 registerButton.setOnClickListener(new View.OnClickListener(){  
 @Override  
 public void onClick(View v){  
 Intent intent = new Intent(MainActivity.this,RegisterActivity.class); //Redirect the customer to the registration page:  
 startActivity(intent);  
 }  
 });  
  
 /\*Navigating to the Login Page After Clicking the Login Button\*/  
 Button loginButton=findViewById(R.id.*home\_login\_button*);  
 loginButton.setOnClickListener(new View.OnClickListener(){  
 @Override  
 public void onClick(View v){  
 Intent intent = new Intent(MainActivity.this,LoginActivity.class); //Redirect the customer to the login page:  
 startActivity(intent);  
 }  
 });  
 }  
}

The **MainActivity.java** code does the following:

* Creates the main screen (this is **activity\_main.xml**) when opening the mobile application.
* From this screen the user can register for an account or login with an existing account.
* If the user is already logged in, it will automatically redirect them to the Home screen (which is the **LandingActivity.java**) of the mobile application.

### User Interface – Registration

The registration screen (which is the **RegisterActivity.java** file) enables the user to register an account.

The user will need to fill in all fields of the registration form, the form consists of:

* First name input
* Last name input
* Email address input
* Password input
* Confirm password input
* Confirm button

If the user is successfully registered, their registration details will be stored in the local database. Afterwards, an alert message will be displayed notifying the user that their registration was successful and they will then be redirected to the login screen (which is **LoginActivity.java file**).

The first iteration of the registration screen and the current iteration are shown below:

##### Iteration 1

A screen shot of a phone

Description automatically generated with low confidence

*Figure 4 - Registration screen for users to create an account.*

##### Iteration 2

A screen shot of a cell phone

Description automatically generated with low confidence

*Figure 5 – Registration screen for users to create an account. The mobile application is running on the Emulator Pixel 6 Pro API 30.*

**Code Snippet:**

package com.example.sustainablespoonfulapp;  
  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.ContentValues;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.database.sqlite.SQLiteDatabase;  
import android.widget.EditText;  
import android.widget.Toast;  
import android.content.Intent;  
import android.database.Cursor;  
  
public class RegisterActivity extends AppCompatActivity {  
  
 //Declaring the database helper variable:  
 private DatabaseHelper databaseHelper;  
 //Declaring variables for all the inputs in the registration form:  
 private EditText nameEditText, surnameEditText, emailEditText, passwordEditText, confirmPasswordEditText;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_register*); //Create the register page:  
  
 //Creating an instance of the DatabaseHelper class:  
 databaseHelper = new DatabaseHelper(this);  
 nameEditText = findViewById(R.id.*register\_name\_text*);  
 surnameEditText = findViewById(R.id.*register\_surname\_text*);  
 emailEditText = findViewById(R.id.*register\_email\_address\_text*);  
 passwordEditText = findViewById(R.id.*register\_password\_text*);  
 confirmPasswordEditText = findViewById(R.id.*register\_confirm\_password\_text*);  
  
 //Inserting the customers' details when pressing the confirm button in the registration form:  
 Button confirmButton = findViewById(R.id.*register\_confirm\_button*);  
 confirmButton.setOnClickListener(new View.OnClickListener(){  
 @Override  
 public void onClick(View v){  
 insertCustomer();  
 }  
 });  
  
 }  
  
 //Function to insert customer details into the customer table:  
 private void insertCustomer(){  
 //Getting all of the data that was input in the form:  
 String name = nameEditText.getText().toString().trim();  
 String surname = surnameEditText.getText().toString().trim();  
 String email = emailEditText.getText().toString().trim();  
 String password = passwordEditText.getText().toString().trim();  
 String confirmPassword = confirmPasswordEditText.getText().toString().trim();  
  
 //Check if any of the input fields are empty before inserting the customer details:  
 if(name.isEmpty()||surname.isEmpty()||email.isEmpty()||password.isEmpty()||confirmPassword.isEmpty()){  
 //Display a message prompting customers to enter their details on the page:  
 Toast.*makeText*(RegisterActivity.this, "Please fill out all fields in this form.", Toast.*LENGTH\_SHORT*).show();  
 return; //Exiting the method early:  
 }  
  
 //If the password and confirm password match insert the details into the customer table:  
 if(password.equals(confirmPassword)){  
 //Getting a writable database:  
 SQLiteDatabase db = databaseHelper.getWritableDatabase();  
  
 //Checking if the email address already exists in the customer table:  
 boolean emailExists = checkEmailExists(email,db);  
  
 if(emailExists){  
 Toast.*makeText*(RegisterActivity.this, "This email address already exists! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }else{ //Does not exist so insert details into the customer table:  
 ContentValues values = new ContentValues();  
 values.put(DatabaseHelper.*COLUMN\_CUSTOMER\_NAME*, name);  
 values.put(DatabaseHelper.*COLUMN\_CUSTOMER\_SURNAME*, surname);  
 values.put(DatabaseHelper.*COLUMN\_CUSTOMER\_EMAIL*,email);  
 values.put(DatabaseHelper.*COLUMN\_CUSTOMER\_PASSWORD*, password);  
  
 long rowID = db.insert(DatabaseHelper.*TABLE\_NAME\_CUSTOMER*,null,values);  
  
 //Closing the database after inserting the customer's details:  
 db.close();  
  
 //If the row ID is not equal to minus one, display a success message:  
 if(rowID != -1){  
 //Display a message to the customer notifying them that their registration was successful:  
 Toast.*makeText*(RegisterActivity.this, "Registration was successful!", Toast.*LENGTH\_SHORT*).show();  
 startActivity(new Intent(RegisterActivity.this, LoginActivity.class)); //Redirect the customer to the login page:  
 finish(); //Finishing the current activity so that customers' cannot go back to it when pressing the back button:  
  
 }else{  
 //Displaying a message at the bottom notifying the customer that their registration has failed:  
 Toast.*makeText*(RegisterActivity.this, "Registration failed! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 }else{ //Passwords do not match, display a message at the bottom notifying the customer that their passwords do not match:  
 Toast.*makeText*(RegisterActivity.this, "The passwords entered do not match! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
  
 private boolean checkEmailExists(String email, SQLiteDatabase db){  
 //Define which column to retrieve from the database:  
 String[] projection = {databaseHelper.getColumnEmail()};  
 //Select any email addresses that are in the customer table that match the email address entered:  
 String selection = databaseHelper.getColumnEmail() + " = ?";  
 //Specify the argument for the query, this will be email:  
 String[] selectionArgs = {email};  
 //Query the customer table for any matching records:  
 Cursor cursor = db.query(DatabaseHelper.*TABLE\_NAME\_CUSTOMER*, projection, selection, selectionArgs, null, null, null);  
 //Checking if any records were found/records greater than 0:  
 boolean exists = cursor.getCount()>0;  
 //Close the cursor so that associated resources can be released:  
 cursor.close();  
 //Return the result (if the email exists or not):  
 return exists;  
 }  
}

The **RegisterActivity.java** code does the following:

* Creates the registration screen (which is **activity\_register.xml**).
* Creates an instance of the **DatabaseHelper** class.
* Checks if the email address already exists in the database and displays an alert message warning the user to try entering another email address.
* Checks if the password and confirm password textboxes match, if they do not match it display an alert message warning the user to check the passwords that they have entered.
* After all the checks have been done, the users’ details are inserted into the local database.

### User Interface – Login

The login screen (which is the **LoginActivity.java** file) allows the user to enter their login credentials.

The login form consists of:

* Email address input
* Password input
* Confirm button

The email address and password need to match the email address and password stored in the local database for the user to gain access to the mobile applications’ home page. Once the user has been verified, they will be redirected to the landing page (which is the **LandingActivity.java** file). An alert message will be displayed on the screen notifying the user that they have been successfully logged in.

The first iteration of the login screen and the current iteration are shown below:

##### Iteration 1

A screen shot of a phone

Description automatically generated with medium confidence

*Figure 7 - Login Screen so that users can login using their existing credentials.*

##### Iteration 2

A screen shot of a cell phone

Description automatically generated with low confidence

*Figure 8 – Login Screen so that users can login using their existing credentials. The mobile application is running on the Emulator Pixel 6 Pro API 30.*

**Code Snippet:**

package com.example.sustainablespoonfulapp;  
  
import android.os.Bundle;  
import android.content.Intent;  
import android.database.Cursor;  
import android.database.sqlite.SQLiteDatabase;  
import android.view.View;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.Toast;  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.SharedPreferences;  
import android.preference.PreferenceManager;  
  
public class LoginActivity extends AppCompatActivity {  
  
 //Declaring the database helper variable:  
 private DatabaseHelper databaseHelper;  
  
 //Declaring variables for the email and password inputs:  
 private EditText emailEditText, passwordEditText;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_login*); //Create the login page:  
  
 //Creating an instance of the DatabaseHelper class:  
 databaseHelper = new DatabaseHelper(this);  
 emailEditText = findViewById(R.id.*login\_email\_address\_text*);  
 passwordEditText = findViewById(R.id.*login\_password\_text*);  
  
 //Setting up the login button click listener:  
 Button loginButton = findViewById(R.id.*login\_confirm\_button*);  
  
 loginButton.setOnClickListener(new View.OnClickListener(){  
 @Override  
 public void onClick(View v){  
 login();  
 }  
 });  
 }  
  
 private void login(){  
 //Getting the email and password that the customer has entered and trimming it:  
 String email = emailEditText.getText().toString().trim();  
 String password = passwordEditText.getText().toString().trim();  
  
 //Checking if any of the input fields are empty before logging in the customer:  
 if(email.isEmpty()||password.isEmpty()){  
 //Display a message to the customer asking them to fill their details in on the form:  
 Toast.*makeText*(LoginActivity.this, "Please fill out all fields in this form.", Toast.*LENGTH\_SHORT*).show();  
 return; //Exit the method early:  
 }  
  
 //Getting a readable database:  
 SQLiteDatabase db = databaseHelper.getReadableDatabase();  
  
 //Checking if the email and password match a record in the customer table:  
 boolean loginSuccess = checkEmailAndPassword(email, password, db);  
  
 //If a record was found and it matches/login was successful display a message and redirect to the landing page:  
 if(loginSuccess){  
 //Store the customer details in Shared Preferences:  
 SharedPreferences sharedPreferences = PreferenceManager.*getDefaultSharedPreferences*(getApplicationContext());  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("email",email); //Store the customers email:  
 editor.putBoolean("isLoggedIn",true); //Store the customers' login status:  
 editor.apply(); //Save the changes to Shared Preferences:  
  
 //Displaying a success message to the customer notifying them that they have been logged in:  
 Toast.*makeText*(LoginActivity.this, "Logged in successfully!", Toast.*LENGTH\_SHORT*).show();  
 //Redirecting to the home Page:  
 startActivity(new Intent(LoginActivity.this, LandingActivity.class));  
 finish(); //Finishing the current activity so that users cannot go back to it when pressing the back button:  
 }else{ //Password or email was incorrect:  
 //Display an error message to the customer:  
 Toast.*makeText*(LoginActivity.this, "Invalid email or password! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }  
 //Closing the database:  
 db.close();  
 }  
  
 private boolean checkEmailAndPassword(String email, String password, SQLiteDatabase db){  
 //Define which columns to retrieve from the customer table in the database:  
 String[] projection = {databaseHelper.getColumnEmail()};  
 //Select any email addresses and passwords that are in the customer table that match the email address and password entered:  
 String selection = databaseHelper.getColumnEmail() + " = ? AND " + databaseHelper.getColumnPassword() + " = ?";  
 //Specify the arguments for the query, this will be email and password:  
 String[] selectionArgs = {email, password};  
 //Query the customer table for any matching records:  
 Cursor cursor = db.query(DatabaseHelper.*TABLE\_NAME\_CUSTOMER*, projection, selection, selectionArgs, null, null, null);  
 //Checking if any records were found/records greater than 0:  
 boolean exists = cursor.getCount()>0;  
 //Close the cursor so that associated resources can be released:  
 cursor.close();  
 //Return the result (if the email exists or not):  
 return exists;  
 }  
}

The **LoginActivity.java** code does the following:

* Creates the login screen (which is the **activity\_login.xml**).
* Created an instance of the **DatabaseHelper** class.
* If the email and password textboxes are left empty, it displays an alert warning the user to fill in their fields.
* It checks if the email address and password entered matches the details in the customer table.
* If the details match, an alert message is displayed on screen notifying the user that they have been logged in. Afterwards the user is redirected to the landing screen (which is the **LandingActivity.java file**).

### User Interface – Retail Partners

The first iteration of the landing screen and the current iteration are shown below:

##### Iteration 1

A screen shot of a phone

Description automatically generated with medium confidence

*Figure 10 – Landing screen for the Sustainable Spoonful Mobile Application.*

##### Iteration 2

A screen shot of a cell phone

Description automatically generated with medium confidence

*Figure 11 – Landing screen for the Sustainable Spoonful Mobile Application. The mobile application is running on the Emulator Pixel 6 Pro API 30.*

This is the landing/home screen for the mobile application. This screen is only displayed once the user has successfully logged in. On this screen users can navigate to other pages (such as the Discounted Products and Account page) using the bottom navigation menu.

**Code Snippet:**

package com.example.sustainablespoonfulapp;  
  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.preference.PreferenceManager;  
import android.view.MenuItem;  
import android.widget.ImageButton;  
import android.widget.Toast;  
import androidx.annotation.NonNull;  
import androidx.appcompat.app.AppCompatActivity;  
import com.google.android.material.bottomnavigation.BottomNavigationView;  
import com.google.android.material.navigation.NavigationBarView;  
import android.view.View;  
  
public class LandingActivity extends AppCompatActivity {  
  
 BottomNavigationView bottom\_nav\_bar;  
  
 ImageButton landing\_picknpay\_button, landing\_foodloversmarket\_button, landing\_checkers\_button, landing\_woolworths\_button;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_landing*); //Create the home page:  
  
 bottom\_nav\_bar = findViewById(R.id.*bottom\_nav\_bar*);  
 bottom\_nav\_bar.setSelectedItemId(R.id.*home\_bottom\_navigation*); //Set the home icon to selected when on this page:  
  
 bottom\_nav\_bar.setOnItemSelectedListener(new NavigationBarView.OnItemSelectedListener() {  
 @Override  
 public boolean onNavigationItemSelected(@NonNull MenuItem item) {  
 //Create a switch statement:  
 switch (item.getItemId()){  
 //If the home icon is clicked, stay on the home page:  
 case R.id.*home\_bottom\_navigation*:  
 return true;  
 //If the search icon is clicked, go to the search discounts page:  
 case R.id.*search\_bottom\_navigation*:  
 startActivity(new Intent(LandingActivity.this,DiscountActivity.class)); //Redirect the customer to the search discount page:  
 return true;  
 //If the account icon is clicked, go to the account page:  
 case R.id.*account\_bottom\_navigation*:  
 startActivity(new Intent(LandingActivity.this,AccountActivity.class)); //Redirect the customer to the account page:  
 return true;  
 default:  
 return false;  
 }  
 }  
 });  
  
 //For the image buttons on the landing screen:  
 landing\_picknpay\_button = findViewById(R.id.*landing\_picknpay\_button*);  
 landing\_foodloversmarket\_button = findViewById(R.id.*landing\_foodloversmarket\_button*);  
 landing\_checkers\_button = findViewById(R.id.*landing\_checkers\_button*);  
 landing\_woolworths\_button = findViewById(R.id.*landing\_woolworths\_button*);  
  
 landing\_picknpay\_button.setOnClickListener(new View.OnClickListener(){  
 public void onClick(View v){  
 Intent intent = new Intent(LandingActivity.this,PicknPayStoreActivity.class);  
 startActivity(intent);  
 }  
 });  
  
 landing\_foodloversmarket\_button.setOnClickListener(new View.OnClickListener(){  
 public void onClick(View v){  
 Intent intent = new Intent(LandingActivity.this,FoodLoversStoreActivity.class);  
 startActivity(intent);  
 }  
 });  
  
 landing\_checkers\_button.setOnClickListener(new View.OnClickListener(){  
 public void onClick(View v){  
 Intent intent = new Intent(LandingActivity.this,CheckersStoreActivity.class);  
 startActivity(intent);  
 }  
 });  
  
 landing\_woolworths\_button.setOnClickListener(new View.OnClickListener(){  
 public void onClick(View v){  
 Intent intent = new Intent(LandingActivity.this,WoolworthsStoreActivity.class);  
 startActivity(intent);  
 }  
 });  
 }  
  
 @Override  
 protected void onResume() {  
 super.onResume();  
  
 //Checking if the customer is logged in:  
 SharedPreferences sharedPreferences = PreferenceManager.*getDefaultSharedPreferences*(getApplicationContext());  
 boolean isLoggedIn = sharedPreferences.getBoolean("isLoggedIn",false); //Set is logged in to true:  
  
 //If the customer is not logged in, redirect to the home page:  
 if(!isLoggedIn) {  
 Toast.*makeText*(LandingActivity.this, "Please log in to continue!", Toast.*LENGTH\_SHORT*).show(); //Display a message to the customer asking them to log in:  
 startActivity(new Intent(LandingActivity.this,MainActivity.class)); //Redirect to the main page of the application:  
 finish(); //Finishing the current activity so that customers' cannot go back to it when pressing the back button:  
 }  
 }  
}

The **LandingActivity.java** code does the following:

* Creates the home screen (which is **activity\_landing.xml**).
* Setups up the bottom navigation bar, the three icons do the following:
  + The home icon will leave the user on the home screen.
  + The search icon will redirect the user to the Discounts Screen
  + The user icon will redirect the user to the accounts screen.
* Loads the images for the stores from the database.
* Makes the images clickable and enables users to navigate to each of the store’s pages which show an image of the store, their location, and a small description.
* The **onResume()** function has the purpose of keeping the user logged in even after they have closed the mobile application.

### User Interface – Discounts

This is the discount screen (which is the **DiscountActivity.java** file) of the mobile application. On this screen users can view the different stores as well as how many discounts they have available currently. From here users can select a store by pressing on the store card. They will then be redirected to the discounted products screen.

The first iteration of the discounts screen and the current iteration are shown below:

##### Iteration 1

A cell phone with text on the screen

Description automatically generated with low confidence

*Figure 13 - Discounts Screen for users to search for discounted products based on the store and location.*

##### Iteration 2

A picture containing text, screenshot, mobile phone, gadget

Description automatically generated

*Figure 14 – Discounts Screen for users to search for discounted products based on the store and location. The mobile application is running on the Emulator Pixel 6 Pro API 30.*

Code Snippet:

package com.example.sustainablespoonfulapp;  
  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.preference.PreferenceManager;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.Toast;  
import android.widget.TextView;  
import androidx.annotation.NonNull;  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.cardview.widget.CardView;  
import com.google.android.material.bottomnavigation.BottomNavigationView;  
import com.google.android.material.navigation.NavigationBarView;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.widget.ImageView;  
import java.util.ArrayList;  
import java.util.List;  
  
public class DiscountActivity extends AppCompatActivity {  
  
 // Create a list for the store images:  
 private List<Bitmap> storeImages;  
  
 // TextViews used to display the number of discounts that are available on the page:  
 TextView picknpayTextView;  
 TextView checkersTextView;  
 TextView woolworthsTextView;  
 TextView foodloversmarketTextView;  
 BottomNavigationView bottom\_nav\_bar;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_discount*);  
  
 // Initializing the store discount images:  
 storeImages = new ArrayList<>();  
  
 // Fetch the store images from the database:  
 DatabaseHelper databaseHelper = new DatabaseHelper(this);  
 List<byte[]> imageBytes = databaseHelper.getAllStoreImages();  
  
 // Convert the byte arrays to bitmaps:  
 for (byte[] bytes : imageBytes) {  
 Bitmap bitmap = BitmapFactory.*decodeByteArray*(bytes, 0, bytes.length);  
 storeImages.add(bitmap);  
 }  
  
 // Displaying the images in the ImageViews:  
 ImageView store\_image\_view1 = findViewById(R.id.*store\_image\_view1*);  
 ImageView store\_image\_view2 = findViewById(R.id.*store\_image\_view2*);  
 ImageView store\_image\_view3 = findViewById(R.id.*store\_image\_view3*);  
 ImageView store\_image\_view4 = findViewById(R.id.*store\_image\_view4*);  
  
 if (storeImages.size() > 0) { // Display PicknPay logo:  
 store\_image\_view1.setImageBitmap(storeImages.get(0));  
 }  
 if (storeImages.size() > 1) { // Display Woolworths logo:  
 store\_image\_view2.setImageBitmap(storeImages.get(1));  
 }  
 if (storeImages.size() > 2) { // Display Checkers logo:  
 store\_image\_view3.setImageBitmap(storeImages.get(2));  
 }  
 if (storeImages.size() > 3) { // Display Foodloversmarket logo:  
 store\_image\_view4.setImageBitmap(storeImages.get(3));  
 }  
  
 // This will be used to find the location of the TextViews on the discounts page:  
 picknpayTextView = findViewById(R.id.*picknpay\_card\_text*);  
 woolworthsTextView = findViewById(R.id.*woolworths\_card\_text*);  
 checkersTextView = findViewById(R.id.*checkers\_card\_text*);  
 foodloversmarketTextView = findViewById(R.id.*foodloversmarket\_card\_text*);  
  
 // Create a new database helper:  
 DatabaseHelper dbhelper = new DatabaseHelper(this);  
  
 // Set the retailer ids for each store that will be passed into the sql count discounted products query:  
 int picknpayId = 1, woolworthsId = 2, checkersId = 3, foodloversID = 4;  
  
 // Set the text for each TextView by calling the getDiscountProductCount function and passing in the retailer id:  
 picknpayTextView.setText(dbhelper.getDiscountedProductCount(picknpayId));  
 woolworthsTextView.setText(dbhelper.getDiscountedProductCount(woolworthsId));  
 checkersTextView.setText(dbhelper.getDiscountedProductCount(checkersId));  
 foodloversmarketTextView.setText(dbhelper.getDiscountedProductCount(foodloversID));  
  
 // Set click listeners for the card views:  
 CardView cardView1 = findViewById(R.id.*cardPicknpay*);  
 CardView cardView2 = findViewById(R.id.*cardCheckers*);  
 CardView cardView3 = findViewById(R.id.*cardWoolworths*);  
 CardView cardView4 = findViewById(R.id.*cardFoodLoversMarket*);  
  
 cardView1.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 startProductViewActivity(1);  
 }  
 });  
  
 cardView2.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 startProductViewActivity(2);  
 }  
 });  
  
 cardView3.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 startProductViewActivity(3);  
 }  
 });  
  
 cardView4.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 startProductViewActivity(4);  
 }  
 });  
  
 bottom\_nav\_bar = findViewById(R.id.*bottom\_nav\_bar*);  
 bottom\_nav\_bar.setSelectedItemId(R.id.*search\_bottom\_navigation*); // Set the search icon to selected when on this page:  
  
 bottom\_nav\_bar.setOnItemSelectedListener(new NavigationBarView.OnItemSelectedListener() {  
 @Override  
 public boolean onNavigationItemSelected(@NonNull MenuItem item) {  
 // Create a switch statement:  
 switch (item.getItemId()) {  
 // If the home icon is clicked, go to the home page:  
 case R.id.*home\_bottom\_navigation*:  
 startActivity(new Intent(DiscountActivity.this, LandingActivity.class)); // Redirect the customer to the home page:  
 return true;  
 // If the search icon is clicked, stay on the search discounts page:  
 case R.id.*search\_bottom\_navigation*:  
 startActivity(new Intent(DiscountActivity.this, DiscountActivity.class)); // Stay on the search discount page:  
 return true;  
 // If the account icon is clicked, go to the account page:  
 case R.id.*account\_bottom\_navigation*:  
 startActivity(new Intent(DiscountActivity.this, AccountActivity.class)); // Redirect the customer to the account page:  
 return true;  
 default:  
 return false;  
 }  
 }  
 });  
 }  
  
 private void startProductViewActivity(int retailerId) {  
 // Start the ProductViewActivity passing the retailer ID as an extra:  
 Intent intent = new Intent(DiscountActivity.this, ProductViewActivity.class);  
 intent.putExtra("retailerId", retailerId);  
 startActivity(intent);  
 }  
 @Override  
 protected void onResume() {  
 super.onResume();  
  
 // Checking if the customer is logged in:  
 SharedPreferences sharedPreferences = PreferenceManager.*getDefaultSharedPreferences*(getApplicationContext());  
 boolean isLoggedIn = sharedPreferences.getBoolean("isLoggedIn", false); // Set is logged in to true:  
  
 // If the customer is not logged in, redirect to the home page:  
 if (!isLoggedIn) {  
 Toast.*makeText*(DiscountActivity.this, "Please log in to continue!", Toast.*LENGTH\_SHORT*).show(); // Display a message to the customer asking them to log in:  
 startActivity(new Intent(DiscountActivity.this, MainActivity.class)); // Redirect to the main page of the application:  
 finish(); // Finishing the current activity so that customers' cannot go back to it when pressing the back button:  
 }  
 }  
}

The **DiscountActivity.java** code does the following:

* Changes the search icon to a different colour to show users that they are on this page.
* Creates a switch statement that:
  + Checks if the home icon has been pressed, if it has, redirect to the home screen.
  + If the search icon is pressed, stay on the discounts screen.
  + If the account icon has been pressed, redirect to account screen.
* Images are pulled from the database and are converted from bitmap format to a drawable file format.
* The cards are clickable and navigate to each of the stores discounted products pages (which is the **ProductViewActivity.java** file).

### User Interface – Products

The first iteration of the discounted products screen (which is the **ProductViewActivity.java** file) and the current iteration are shown below:

##### Iteration 1

A picture containing screenshot, mobile phone, gadget, communication device

Description automatically generated

*Figure 16 – Discounted products screen for each store.*

##### Iteration 2

A picture containing text, mobile phone, screenshot, communication device

Description automatically generated

*Figure 17 Discounted products screen for each store. The mobile application is running on the Emulator Pixel 6 Pro API 30*

This is the products screen (which is the ProductViewActivity.java file). Here users can view the discounted products on offer by a particular store and view the discount codes for the products on offer.

**Code Snippet:**

package com.example.sustainablespoonfulapp;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.MenuItem;  
  
import androidx.annotation.NonNull;  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.recyclerview.widget.LinearLayoutManager;  
import androidx.recyclerview.widget.RecyclerView;  
  
import com.google.android.material.bottomnavigation.BottomNavigationView;  
import com.google.android.material.navigation.NavigationBarView;  
  
import java.util.List;  
  
public class ProductViewActivity extends AppCompatActivity {  
 BottomNavigationView bottom\_nav\_bar;  
  
 private String getRetailerName(int retailerId) {  
 // Retrieve the retailer name based on the retailer ID from the database:  
 DatabaseHelper databaseHelper = new DatabaseHelper(this);  
 return databaseHelper.getRetailerNameById(retailerId);  
 }  
  
 private List<DiscountedProduct> getProductsByRetailer(int retailerId) {  
 // Retrieve the list of products for the specified retailer from the database:  
 DatabaseHelper databaseHelper = new DatabaseHelper(this);  
 return databaseHelper.getDiscountedProductsByRetailerId(retailerId);  
 }  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*recycler\_view*);  
  
 // Retrieve the retailer ID passed from the previous activity:  
 int retailerId = getIntent().getIntExtra("retailerId", -1);  
  
 // Set the title of the activity based on the retailer:  
 String retailerName = getRetailerName(retailerId);  
 setTitle(retailerName + " Discounted Items");  
  
 // Set up the RecyclerView:  
 RecyclerView recyclerView = findViewById(R.id.*discounted*);  
  
 ProductAdapter productAdapter = new ProductAdapter(getProductsByRetailer(retailerId), this);  
 recyclerView.setAdapter(productAdapter);  
 recyclerView.setLayoutManager(new LinearLayoutManager(this));  
  
 }  
}

The **ProductViewActivity.java** code does the following:

* It utilizes a recycler view that pulls the product images from the database and displays them in the product view.
* It uses a constructor file (which is the **DiscountedProduct.java**) and an adapter file (which is the **ProductAdapter.java**).

### User Interface – Account

The first iteration of the account screen (which is the AccountActivity.java file) and the current iteration are shown below:

##### Iteration 1

A screen shot of a cell phone

Description automatically generated with medium confidence

*Figure 18 –Account screen*

##### Iteration 2

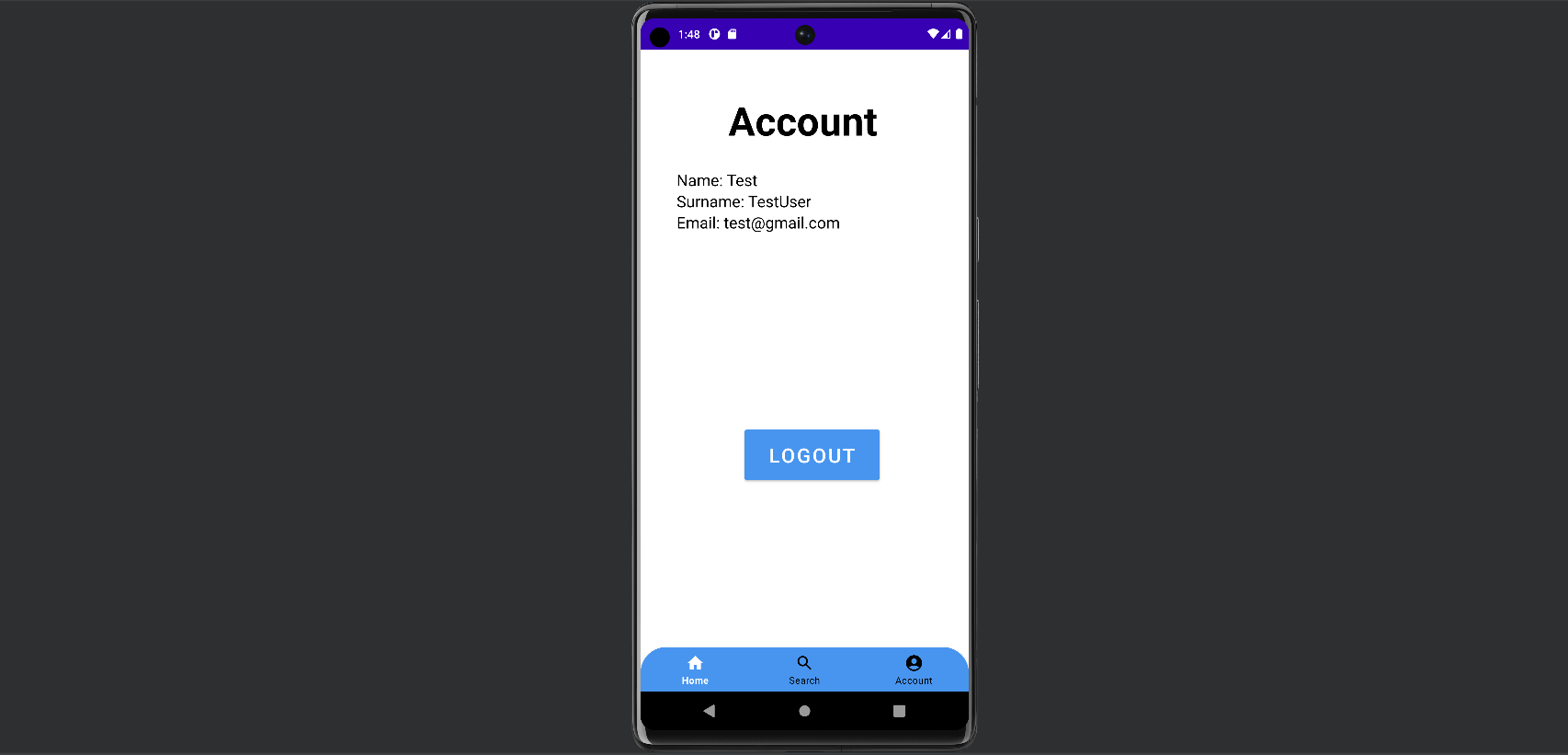


Figure 19 – Account screen. The mobile application is running on the *Emulator Pixel 6 Pro API 30.*

This is the user accounts screen.

The account form consists of:

* The users name display.
* The users surname display.
* The users email display.
* Logout button.

The account section will display the name, surname, and email address of the logged in user. The user can also logout here by clicking the logout button.

**Code Snippet:**

package com.example.sustainablespoonfulapp;  
  
import androidx.annotation.NonNull;  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.preference.PreferenceManager;  
import android.view.MenuItem;  
import android.widget.Toast;  
import com.google.android.material.bottomnavigation.BottomNavigationView;  
import com.google.android.material.navigation.NavigationBarView;  
import androidx.appcompat.app.AlertDialog;  
import android.content.DialogInterface;  
import android.widget.Button;  
import android.database.Cursor;  
import android.database.sqlite.SQLiteDatabase;  
import android.widget.TextView;  
  
  
public class AccountActivity extends AppCompatActivity{  
  
 BottomNavigationView bottom\_nav\_bar;  
 Button logoutButton;  
  
 TextView nameTextView;  
 TextView surnameTextView;  
 TextView emailTextView;  
  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_account*); //Create the account page:  
  
 //Checking if the customer is logged in:  
 SharedPreferences sharedPreferences = PreferenceManager.*getDefaultSharedPreferences*(getApplicationContext());  
 String email = sharedPreferences.getString("email","");  
  
 //If the customer is not logged in, display a message and redirect to the main page when opening the application:  
 if(email.isEmpty()){  
 Toast.*makeText*(AccountActivity.this, "Please log in to continue!", Toast.*LENGTH\_SHORT*).show(); //Display a message to the customer asking them to log in:  
 startActivity(new Intent(AccountActivity.this, MainActivity.class)); //Redirect the customer to the main page when opening the application:  
 finish(); //Finishing the current activity so that customers' cannot go back to it when pressing the back button:  
 return; //Return early so that the rest of the method is not executed:  
 }  
  
 bottom\_nav\_bar = findViewById(R.id.*bottom\_nav\_bar*);  
 bottom\_nav\_bar.setSelectedItemId(R.id.*account\_bottom\_navigation*); //Set the account icon to selected when on this page:  
 logoutButton = findViewById(R.id.*account\_logout\_button*);  
 nameTextView = findViewById(R.id.*account\_name\_text*);  
 surnameTextView = findViewById(R.id.*account\_surname\_text*);  
 emailTextView = findViewById(R.id.*account\_email\_text*);  
  
 bottom\_nav\_bar.setOnItemSelectedListener(new NavigationBarView.OnItemSelectedListener() {  
 @Override  
 public boolean onNavigationItemSelected(@NonNull MenuItem item) {  
 //Create a switch statement:  
 switch (item.getItemId()){  
 //If the home icon is clicked, go to the home page:  
 case R.id.*home\_bottom\_navigation*:  
 startActivity(new Intent(AccountActivity.this,LandingActivity.class)); //Redirect the customer to the home page:  
 finish(); //Finishing the current activity so that customers' cannot go back to it when pressing the back button:  
 return true;  
 //If the search icon is clicked,go to the search discounts page:  
 case R.id.*search\_bottom\_navigation*:  
 startActivity(new Intent(AccountActivity.this,DiscountActivity.class)); //Redirect the customer to the search discount page:  
 finish(); //Finishing the current activity so that customers' cannot go back to it when pressing the back button:  
 return true;  
 //If the account icon is clicked, stay on the account page:  
 case R.id.*account\_bottom\_navigation*:  
 return true;  
 default:  
 return false;  
 }  
 }  
 });  
  
 //When the logout button is clicked, call the showLogoutConfirmationBox() function:  
 logoutButton.setOnClickListener(v -> {  
 showLogoutConfirmationBox();  
 });  
  
 loadCustomerDetails();  
 }  
  
 private void loadCustomerDetails(){  
 SharedPreferences sharedPreferences = PreferenceManager.*getDefaultSharedPreferences*(getApplicationContext());  
 String email = sharedPreferences.getString("email", "");  
  
 DatabaseHelper databaseHelper = new DatabaseHelper(this);  
 SQLiteDatabase db = databaseHelper.getReadableDatabase();  
  
 String[] projection = {  
 DatabaseHelper.*COLUMN\_CUSTOMER\_NAME*,  
 DatabaseHelper.*COLUMN\_CUSTOMER\_SURNAME*,  
 DatabaseHelper.*COLUMN\_CUSTOMER\_EMAIL* };  
  
 String selection = DatabaseHelper.*COLUMN\_CUSTOMER\_EMAIL* + " =?";  
 String[] selectionArgs = {email};  
  
 Cursor cursor = db.query(  
 DatabaseHelper.*TABLE\_NAME\_CUSTOMER*,  
 projection,  
 selection,  
 selectionArgs,  
 null,  
 null,  
 null  
 );  
  
 if(cursor != null && cursor.moveToFirst()){  
 int customerNameIndex = cursor.getColumnIndex(DatabaseHelper.*COLUMN\_CUSTOMER\_NAME*);  
 int customerSurnameIndex = cursor.getColumnIndex(DatabaseHelper.*COLUMN\_CUSTOMER\_SURNAME*);  
 int customerEmailIndex = cursor.getColumnIndex(DatabaseHelper.*COLUMN\_CUSTOMER\_EMAIL*);  
  
  
 String customerName = cursor.getString(customerNameIndex);  
 String customerSurname = cursor.getString(customerSurnameIndex);  
 String customerEmail = cursor.getString(customerEmailIndex);  
  
 cursor.close();  
 db.close();  
  
 nameTextView.setText("Name: " + customerName);  
 surnameTextView.setText("Surname: " + customerSurname);  
 emailTextView.setText("Email: " + customerEmail);  
 }  
 }  
  
 //Function to display a confirmation box to the customer asking if they want to logout:  
 private void showLogoutConfirmationBox(){  
 AlertDialog.Builder builder = new AlertDialog.Builder(this);  
 builder.setTitle("Logout") //Title of confirmation box:  
 .setMessage("Are you sure that you want to logout?") //Message in confirmation box:  
 .setPositiveButton("Yes", new DialogInterface.OnClickListener() { //Option 1: Customer clicks yes, call the logoutCustomer function:  
 @Override  
 public void onClick(DialogInterface dialogInterface, int i) {  
 //Log out the customer and redirect to the Home page:  
 logoutCustomer();  
 }  
 })  
 .setNegativeButton("No",null) //Option 2: Customer clicks no:  
 .show();  
 }  
  
 private void logoutCustomer(){  
 //Clear the customers' session:  
 SharedPreferences sharedPreferences = PreferenceManager.*getDefaultSharedPreferences*(getApplicationContext());  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.remove("email"); //Remove the email address that was stored in the session:  
 editor.putBoolean("isLoggedIn",false);  
 editor.apply(); //Save the changes to the SharedPreferences:  
  
 //Display a message to the customer saying that they have logged out, then redirect to the main page when opening the application:  
 Toast.*makeText*(AccountActivity.this, "Logged out successfully!", Toast.*LENGTH\_SHORT*).show(); //Display a message to the customer saying that they have logged out:  
  
 //Redirect to the main page if the activity is active:  
 Intent intent= new Intent(AccountActivity.this, MainActivity.class); //Redirect the customer to the main page when opening the application:  
 intent.setFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TOP*);  
 startActivity(intent);  
 finish(); //Finishing the current activity so that customers' cannot go back to it when pressing the back button:  
 }  
}

This **AccountActivity.java** code does the following:

* Creates the account screen (which is the **activity\_account.xml**).
* Checks if the customer is logged, if not a message will be displayed, and they will be redirected to the Main screen so that they can register and login.
* Sets the account icon in the bottom navigation menu to be selected when on this screen.
* Loads the **loadCustomerDetails()** function which fetches the users’ details based on their email address.

## Business Logic Development

### Algorithm 1 – Login

**Code Snippet:**

package com.example.sustainablespoonfulapp;  
  
import android.os.Bundle;  
import android.content.Intent;  
import android.database.Cursor;  
import android.database.sqlite.SQLiteDatabase;  
import android.view.View;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.Toast;  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.SharedPreferences;  
import android.preference.PreferenceManager;  
  
public class LoginActivity extends AppCompatActivity {  
  
 //Declaring the database helper variable:  
 private DatabaseHelper databaseHelper;  
  
 //Declaring variables for the email and password inputs:  
 private EditText emailEditText, passwordEditText;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_login*); //Create the login page:  
  
 //Creating an instance of the DatabaseHelper class:  
 databaseHelper = new DatabaseHelper(this);  
 emailEditText = findViewById(R.id.*login\_email\_address\_text*);  
 passwordEditText = findViewById(R.id.*login\_password\_text*);  
  
 //Setting up the login button click listener:  
 Button loginButton = findViewById(R.id.*login\_confirm\_button*);  
  
 loginButton.setOnClickListener(new View.OnClickListener(){  
 @Override  
 public void onClick(View v){  
 login();  
 }  
 });  
 }  
  
 private void login(){  
 //Getting the email and password that the customer has entered and trimming it:  
 String email = emailEditText.getText().toString().trim();  
 String password = passwordEditText.getText().toString().trim();  
  
 //Checking if any of the input fields are empty before logging in the customer:  
 if(email.isEmpty()||password.isEmpty()){  
 //Display a message to the customer asking them to fill their details in on the form:  
 Toast.*makeText*(LoginActivity.this, "Please fill out all fields in this form.", Toast.*LENGTH\_SHORT*).show();  
 return; //Exit the method early:  
 }  
  
 //Getting a readable database:  
 SQLiteDatabase db = databaseHelper.getReadableDatabase();  
  
 //Checking if the email and password match a record in the customer table:  
 boolean loginSuccess = checkEmailAndPassword(email, password, db);  
  
 //If a record was found and it matches/login was successful display a message and redirect to the landing page:  
 if(loginSuccess){  
 //Store the customer details in Shared Preferences:  
 SharedPreferences sharedPreferences = PreferenceManager.*getDefaultSharedPreferences*(getApplicationContext());  
 SharedPreferences.Editor editor = sharedPreferences.edit();  
 editor.putString("email",email); //Store the customers email:  
 editor.putBoolean("isLoggedIn",true); //Store the customers' login status:  
 editor.apply(); //Save the changes to Shared Preferences:  
  
 //Displaying a success message to the customer notifying them that they have been logged in:  
 Toast.*makeText*(LoginActivity.this, "Logged in successfully!", Toast.*LENGTH\_SHORT*).show();  
 //Redirecting to the home Page:  
 startActivity(new Intent(LoginActivity.this, LandingActivity.class));  
 finish(); //Finishing the current activity so that users cannot go back to it when pressing the back button:  
 }else{ //Password or email was incorrect:  
 //Display an error message to the customer:  
 Toast.*makeText*(LoginActivity.this, "Invalid email or password! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }  
 //Closing the database:  
 db.close();  
 }  
  
 private boolean checkEmailAndPassword(String email, String password, SQLiteDatabase db){  
 //Define which columns to retrieve from the customer table in the database:  
 String[] projection = {databaseHelper.getColumnEmail()};  
 //Select any email addresses and passwords that are in the customer table that match the email address and password entered:  
 String selection = databaseHelper.getColumnEmail() + " = ? AND " + databaseHelper.getColumnPassword() + " = ?";  
 //Specify the arguments for the query, this will be email and password:  
 String[] selectionArgs = {email, password};  
 //Query the customer table for any matching records:  
 Cursor cursor = db.query(DatabaseHelper.*TABLE\_NAME\_CUSTOMER*, projection, selection, selectionArgs, null, null, null);  
 //Checking if any records were found/records greater than 0:  
 boolean exists = cursor.getCount()>0;  
 //Close the cursor so that associated resources can be released:  
 cursor.close();  
 //Return the result (if the email exists or not):  
 return exists;  
 }  
}

This snippet of code for login does the following:

* Trims the email address and password that the user entered of whitespaces.
* Ensures that none of the input fields have been left blank.
* Fetches the readable database.
* Checks if the user exists in the customer table within the database.
* If the user exists, an alert is displayed, and they are redirected to the home page.
* The session is also stored.
* If the user entered an incorrect email address or password or if they do not exist within the database, an error message will be displayed.

### Algorithm 2 - Registration

**Code Snippet:**

package com.example.sustainablespoonfulapp;  
  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.ContentValues;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.database.sqlite.SQLiteDatabase;  
import android.widget.EditText;  
import android.widget.Toast;  
import android.content.Intent;  
import android.database.Cursor;  
  
public class RegisterActivity extends AppCompatActivity {  
  
 //Declaring the database helper variable:  
 private DatabaseHelper databaseHelper;  
 //Declaring variables for all the inputs in the registration form:  
 private EditText nameEditText, surnameEditText, emailEditText, passwordEditText, confirmPasswordEditText;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_register*); //Create the register page:  
  
 //Creating an instance of the DatabaseHelper class:  
 databaseHelper = new DatabaseHelper(this);  
 nameEditText = findViewById(R.id.*register\_name\_text*);  
 surnameEditText = findViewById(R.id.*register\_surname\_text*);  
 emailEditText = findViewById(R.id.*register\_email\_address\_text*);  
 passwordEditText = findViewById(R.id.*register\_password\_text*);  
 confirmPasswordEditText = findViewById(R.id.*register\_confirm\_password\_text*);  
  
 //Inserting the customers' details when pressing the confirm button in the registration form:  
 Button confirmButton = findViewById(R.id.*register\_confirm\_button*);  
 confirmButton.setOnClickListener(new View.OnClickListener(){  
 @Override  
 public void onClick(View v){  
 insertCustomer();  
 }  
 });  
  
 }  
  
 //Function to insert customer details into the customer table:  
 private void insertCustomer(){  
 //Getting all of the data that was input in the form:  
 String name = nameEditText.getText().toString().trim();  
 String surname = surnameEditText.getText().toString().trim();  
 String email = emailEditText.getText().toString().trim();  
 String password = passwordEditText.getText().toString().trim();  
 String confirmPassword = confirmPasswordEditText.getText().toString().trim();  
  
 //Check if any of the input fields are empty before inserting the customer details:  
 if(name.isEmpty()||surname.isEmpty()||email.isEmpty()||password.isEmpty()||confirmPassword.isEmpty()){  
 //Display a message prompting customers to enter their details on the page:  
 Toast.*makeText*(RegisterActivity.this, "Please fill out all fields in this form.", Toast.*LENGTH\_SHORT*).show();  
 return; //Exiting the method early:  
 }  
  
 //If the password and confirm password match insert the details into the customer table:  
 if(password.equals(confirmPassword)){  
 //Getting a writable database:  
 SQLiteDatabase db = databaseHelper.getWritableDatabase();  
  
 //Checking if the email address already exists in the customer table:  
 boolean emailExists = checkEmailExists(email,db);  
  
 if(emailExists){  
 Toast.*makeText*(RegisterActivity.this, "This email address already exists! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }else{ //Does not exist so insert details into the customer table:  
 ContentValues values = new ContentValues();  
 values.put(DatabaseHelper.*COLUMN\_CUSTOMER\_NAME*, name);  
 values.put(DatabaseHelper.*COLUMN\_CUSTOMER\_SURNAME*, surname);  
 values.put(DatabaseHelper.*COLUMN\_CUSTOMER\_EMAIL*,email);  
 values.put(DatabaseHelper.*COLUMN\_CUSTOMER\_PASSWORD*, password);  
  
 long rowID = db.insert(DatabaseHelper.*TABLE\_NAME\_CUSTOMER*,null,values);  
  
 //Closing the database after inserting the customer's details:  
 db.close();  
  
 //If the row ID is not equal to minus one, display a success message:  
 if(rowID != -1){  
 //Display a message to the customer notifying them that their registration was successful:  
 Toast.*makeText*(RegisterActivity.this, "Registration was successful!", Toast.*LENGTH\_SHORT*).show();  
 startActivity(new Intent(RegisterActivity.this, LoginActivity.class)); //Redirect the customer to the login page:  
 finish(); //Finishing the current activity so that customers' cannot go back to it when pressing the back button:  
  
 }else{  
 //Displaying a message at the bottom notifying the customer that their registration has failed:  
 Toast.*makeText*(RegisterActivity.this, "Registration failed! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 }else{ //Passwords do not match, display a message at the bottom notifying the customer that their passwords do not match:  
 Toast.*makeText*(RegisterActivity.this, "The passwords entered do not match! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
  
 private boolean checkEmailExists(String email, SQLiteDatabase db){  
 //Define which column to retrieve from the database:  
 String[] projection = {databaseHelper.getColumnEmail()};  
 //Select any email addresses that are in the customer table that match the email address entered:  
 String selection = databaseHelper.getColumnEmail() + " = ?";  
 //Specify the argument for the query, this will be email:  
 String[] selectionArgs = {email};  
 //Query the customer table for any matching records:  
 Cursor cursor = db.query(DatabaseHelper.*TABLE\_NAME\_CUSTOMER*, projection, selection, selectionArgs, null, null, null);  
 //Checking if any records were found/records greater than 0:  
 boolean exists = cursor.getCount()>0;  
 //Close the cursor so that associated resources can be released:  
 cursor.close();  
 //Return the result (if the email exists or not):  
 return exists;  
 }  
}

This snippet of code for registration does the following:

* Fetches the user input and removes any whitespaces.
* Checks that none of the fields are empty before inserting into the customer table.
* If any fields are empty, an alert will be displayed notifying the customer to please fill in all the fields.
* Checks to ensure the password entered in the password field and confirm password field match.
* Checks the database to ensure the email address doesn’t already exist.
* Once registration has been successfully completed, an alert will display notifying the user that their registration was successful.
* The user will then be redirected to the login page where they can enter their email address and password.

## Data Access Development

### Database Implementation

For our application we have chosen to work with SQLite.

SQLite is an embedded relational database management system that is widely used in mobile development. It is a lightweight, self-contained, reliable database engine that is available locally and does not require any administration to function. SQLite also comes built into Android Studio making it the ideal choice (SQLite, 2023).

To implement it, the **gradle.build** was included in the file, and a new class called **DatabaseHelper** was created. The tables were then created, and options were provided for it to be create in the other classes when the **onCreate()** function is called. If it has been upgraded since its last use, the table is dropped and recreated again. This will ensure the code remains reusable.

#### Database Creation Class – DatabaseHelper.java

package com.example.sustainablespoonfulapp;  
  
import android.annotation.SuppressLint;  
import android.content.ContentValues;  
import android.content.Context;  
import android.database.sqlite.SQLiteDatabase;  
import android.database.sqlite.SQLiteOpenHelper;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import java.io.ByteArrayOutputStream;  
import java.util.ArrayList;  
import java.util.List;  
import android.database.Cursor;  
  
public class DatabaseHelper extends SQLiteOpenHelper{  
  
 //Providing the database name and version:  
 public static final String *DATABASE\_NAME* = "sustainable\_spoonful.db";  
 public static final int *DATABASE\_VERSION* = 1;  
  
 //Customer Table Constants:  
 public static final String *TABLE\_NAME\_CUSTOMER* = "customer";  
 public static final String *COLUMN\_CUSTOMER\_ID* = "customer\_id";  
 public static final String *COLUMN\_CUSTOMER\_NAME* = "customer\_name";  
 public static final String *COLUMN\_CUSTOMER\_SURNAME* = "customer\_surname";  
 public static final String *COLUMN\_CUSTOMER\_EMAIL* = "customer\_email";  
 public static final String *COLUMN\_CUSTOMER\_PASSWORD* = "customer\_password";  
  
 //Retailer Table Constants:  
 public static final String *TABLE\_NAME\_RETAILER* = "retailer";  
 public static final String *COLUMN\_RETAILER\_ID* = "retailer\_id";  
 public static final String *COLUMN\_RETAILER\_NAME* = "retailer\_name";  
 public static final String *COLUMN\_RETAILER\_ADDRESS* = "address";  
 public static final String *COLUMN\_RETAILER\_IMAGE* = "retailer\_image";  
  
 //Discounted Products Table Constants:  
 public static final String *TABLE\_NAME\_DISCOUNTED\_PRODUCTS* = "discounted\_products";  
 public static final String *COLUMN\_DISCOUNT\_ID* = "discount\_id";  
 public static final String *COLUMN\_DISCOUNT\_CODE* = "discount\_code";  
 public static final String *COLUMN\_DISCOUNT\_RETAILER\_ID* = "retailer\_id";  
 public static final String *COLUMN\_DISCOUNT\_PRODUCT\_NAME* = "product\_name";  
 public static final String *COLUMN\_DISCOUNT\_PERCENTAGE* = "discount\_percentage";  
 public static final String *COLUMN\_DISCOUNT\_IMAGE* = "discount\_image";  
  
 private final Context context; // Add a member variable to store the Context object:  
  
 public DatabaseHelper(Context context){  
 super(context, *DATABASE\_NAME*, null, *DATABASE\_VERSION*);  
 this.context = context; //Assign the context to the member variable:  
 }  
  
 @Override  
 public void onCreate(SQLiteDatabase db){ //Creates database tables and inserts initial data:  
 //Create the database tables:  
 //CUSTOMER TABLE:  
 String createCustomerTable = "CREATE TABLE " + *TABLE\_NAME\_CUSTOMER* + " (" +  
 *COLUMN\_CUSTOMER\_ID* + " INTEGER PRIMARY KEY," +  
 *COLUMN\_CUSTOMER\_NAME* + " TEXT," +  
 *COLUMN\_CUSTOMER\_SURNAME* + " TEXT," +  
 *COLUMN\_CUSTOMER\_EMAIL* + " TEXT," +  
 *COLUMN\_CUSTOMER\_PASSWORD* + " TEXT)";  
 db.execSQL(createCustomerTable);  
  
 //RETAILER TABLE:  
 String createRetailerTable = "CREATE TABLE " + *TABLE\_NAME\_RETAILER* + " (" +  
 *COLUMN\_RETAILER\_ID* + " INTEGER PRIMARY KEY," +  
 *COLUMN\_RETAILER\_NAME* + " TEXT," +  
 *COLUMN\_RETAILER\_ADDRESS* + " TEXT," +  
 *COLUMN\_RETAILER\_IMAGE* + " BLOB)"; //Retailer image column  
 db.execSQL(createRetailerTable);  
  
 //DISCOUNTED PRODUCTS TABLE:  
 String createDiscountedProductsTable = "CREATE TABLE " + *TABLE\_NAME\_DISCOUNTED\_PRODUCTS* + " (" +  
 *COLUMN\_DISCOUNT\_ID* + " INTEGER PRIMARY KEY," +  
 *COLUMN\_DISCOUNT\_CODE* + " TEXT," +  
 *COLUMN\_DISCOUNT\_PERCENTAGE* + " TEXT," +  
 *COLUMN\_DISCOUNT\_PRODUCT\_NAME* + " TEXT," +  
 *COLUMN\_DISCOUNT\_RETAILER\_ID* + " INTEGER," +  
 *COLUMN\_DISCOUNT\_IMAGE* + " BLOB," + //Discounted Product Image Column  
 "FOREIGN KEY (" + *COLUMN\_DISCOUNT\_RETAILER\_ID* + ") REFERENCES " +  
 *TABLE\_NAME\_RETAILER* + "(" + *COLUMN\_RETAILER\_ID* + "))";  
 db.execSQL(createDiscountedProductsTable);  
  
 //Function to insert the data into the Retailer Table:  
 insertRetailerData(db);  
  
 //Function to insert the data into the Discounted Products Table:  
 insertDiscountedProductsData(db);  
 }  
  
 //Function to fetch the discounted products in the discounted products table based on the Retailer ID:  
 public List<DiscountedProduct> getDiscountedProductsByRetailerId(int retailerId) {  
 List<DiscountedProduct> discountedProducts = new ArrayList<>();  
 SQLiteDatabase database = getReadableDatabase();  
  
 String[] columns = {  
 *COLUMN\_DISCOUNT\_RETAILER\_ID*,  
 *COLUMN\_DISCOUNT\_CODE*,  
 *COLUMN\_DISCOUNT\_PERCENTAGE*,  
 *COLUMN\_DISCOUNT\_PRODUCT\_NAME*,  
 *COLUMN\_DISCOUNT\_IMAGE* };  
  
 String selection = *COLUMN\_DISCOUNT\_RETAILER\_ID* + " = ?";  
 String[] selectionArgs = { String.*valueOf*(retailerId) };  
  
 Cursor cursor = database.query(  
 *TABLE\_NAME\_DISCOUNTED\_PRODUCTS*,  
 columns,  
 selection,  
 selectionArgs,  
 null,  
 null,  
 null  
 );  
  
 int columnDiscountCode = cursor.getColumnIndex(*COLUMN\_DISCOUNT\_CODE*);  
 int columnDiscountPercentage = cursor.getColumnIndex(*COLUMN\_DISCOUNT\_PERCENTAGE*);  
 int columnProductName = cursor.getColumnIndex(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*);  
 int columnDiscountImage = cursor.getColumnIndex(*COLUMN\_DISCOUNT\_IMAGE*);  
  
 if (cursor != null && cursor.moveToFirst()) {  
 do {  
 String discountCode = cursor.getString(columnDiscountCode);  
 String discountPercentage = cursor.getString(columnDiscountPercentage);  
 String productName = cursor.getString(columnProductName);  
 byte[] discountImage = cursor.getBlob(columnDiscountImage);  
  
 DiscountedProduct discountedProduct = new DiscountedProduct(discountCode, discountPercentage, productName, discountImage);  
 discountedProducts.add(discountedProduct);  
 } while (cursor.moveToNext());  
 }  
  
 cursor.close();  
 database.close();  
  
 return discountedProducts;  
 }  
  
  
 //Function to count the discounted products in the discounted products table based on the Retailer ID:  
 public String getDiscountedProductCount(int retailerId){  
 //Get a readable database:  
 SQLiteDatabase datab = getReadableDatabase();  
  
 //Create the query to count all of the products in the discounted products table based on the retailer id and return the result:  
 String query = "SELECT COUNT(\*) FROM " + *TABLE\_NAME\_DISCOUNTED\_PRODUCTS* +  
 " WHERE " + *COLUMN\_DISCOUNT\_RETAILER\_ID* + " = " + retailerId;  
 Cursor cursor = datab.rawQuery(query, null);  
  
 //Initialise the counter to 0:  
 int count=0;  
  
 if(cursor !=null && cursor.moveToFirst()){  
 count = cursor.getInt(0);  
 }  
 cursor.close();  
  
 if(count == 0){ //No result was found, display "No Discounts"  
 return "No Discounts";  
 }else{ //Result was found, fetch the number of products and concatenate with a string saying 'Discounts'  
 return count + " Discounts";  
 }  
 }  
  
 //Function to convert drawable resources to bitmap:  
 private byte[] convertImageToByteArray(int imageResource){  
 Bitmap bitmap = BitmapFactory.*decodeResource*(context.getResources(), imageResource);  
 ByteArrayOutputStream stream = new ByteArrayOutputStream();  
 bitmap.compress(Bitmap.CompressFormat.*PNG*, 100, stream);  
 return stream.toByteArray();  
 }  
  
 //Method to retrieve all the store images from the retailer table:  
 public List<byte[]> getAllStoreImages() {  
 List<byte[]> storeImages = new ArrayList<>();  
 SQLiteDatabase database = getReadableDatabase();  
 String query = "SELECT " + *COLUMN\_RETAILER\_IMAGE* + " FROM " + *TABLE\_NAME\_RETAILER*; //query to select all retailer images from the retailer table:  
 Cursor cursor = database.rawQuery(query, null);  
 int columnIndex = cursor.getColumnIndex(*COLUMN\_RETAILER\_IMAGE*);  
  
 if(columnIndex != -1){  
 if(cursor.moveToFirst()){  
 do{  
 byte[] image = cursor.getBlob(columnIndex);  
 storeImages.add(image);  
 }while(cursor.moveToNext());  
 }  
 }  
 cursor.close();  
 return storeImages;  
 }  
  
 @SuppressLint("Range")  
 public String getRetailerNameById(int retailerId) {  
 SQLiteDatabase db = this.getReadableDatabase();  
 String retailerName = "";  
  
 // Query the database to fetch the retailer name based on retailerId  
 String[] projection = {*COLUMN\_RETAILER\_NAME*};  
 String selection = *COLUMN\_RETAILER\_ID* + " = ?";  
 String[] selectionArgs = {String.*valueOf*(retailerId)};  
  
 Cursor cursor = db.query(*TABLE\_NAME\_RETAILER*, projection, selection, selectionArgs, null, null, null);  
  
 if (cursor.moveToFirst() && cursor.getColumnIndex(*COLUMN\_RETAILER\_NAME*) != -1) {  
 retailerName = cursor.getString(cursor.getColumnIndex(*COLUMN\_RETAILER\_NAME*));  
 }  
  
 cursor.close();  
 db.close();  
  
 return retailerName;  
 }  
  
 private void insertRetailerData(SQLiteDatabase db){  
 //Create a ContentValues object so that it can hold the column values for each row:  
 ContentValues values = new ContentValues();  
  
 //Inserting the first shop:  
 values.put(*COLUMN\_RETAILER\_NAME*, "PicknPay");  
 values.put(*COLUMN\_RETAILER\_ADDRESS*, "Greenstone");  
 values.put(*COLUMN\_RETAILER\_IMAGE*, convertImageToByteArray(R.drawable.*picknpay\_discount*));  
 db.insert(*TABLE\_NAME\_RETAILER*, null, values);  
  
 //Inserting the second shop:  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_RETAILER\_NAME*, "Woolworths");  
 values.put(*COLUMN\_RETAILER\_ADDRESS*, "Greenstone");  
 values.put(*COLUMN\_RETAILER\_IMAGE*, convertImageToByteArray(R.drawable.*woolworths\_discount*));  
 db.insert(*TABLE\_NAME\_RETAILER*, null, values);  
  
 //Inserting the third shop:  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_RETAILER\_NAME*, "Checkers");  
 values.put(*COLUMN\_RETAILER\_ADDRESS*, "Meadowdale");  
 values.put(*COLUMN\_RETAILER\_IMAGE*, convertImageToByteArray(R.drawable.*checkers\_discount*));  
 db.insert(*TABLE\_NAME\_RETAILER*, null, values);  
  
 //Inserting the fourth shop:  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_RETAILER\_NAME*, "Food Lover's Market");  
 values.put(*COLUMN\_RETAILER\_ADDRESS*, "Greenstone");  
 values.put(*COLUMN\_RETAILER\_IMAGE*, convertImageToByteArray(R.drawable.*foodloversmarket\_discount*));  
 db.insert(*TABLE\_NAME\_RETAILER*, null, values);  
 }  
  
 public void insertDiscountedProductsData(SQLiteDatabase db){  
 //Create a ContentValues object so that it can hold the column values for each row:  
 ContentValues values = new ContentValues();  
  
 //Insert the first discounted product:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "P1CkNP@yI1");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "10% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "White Bread Loaf");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 1); //PicknPay is the first store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*bread\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "P1CkNP@yO2");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "20% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Oranges 1 KG");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 1); //PicknPay is the first store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*oranges\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "P1CkNP@yS0");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "30% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Sunflower Oil 1 L");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 1); //PicknPay is the first store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*oil\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 //Insert the first discounted product:  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "W0oLWORTHSM3");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "15% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Full Cream Milk 1 L");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 2); //Woolworths is the second store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*milk\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "W0oLWORTHSBB");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "10% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Pack of Bananas");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 2); //Woolworths is the second store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*bananas\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "W0oLWORTHSA@");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "30% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Apples 1.5 KG");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 2); //Woolworths is the second store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*apples\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "CHECKER$U$");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "60% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Ultimate Sandwich");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 3); //Checkers is the third store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*sandwich\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "CHECKER$OO");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "35% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Onions 1 KG");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 3); //Checkers is the third store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*onions\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "CHECKER$AP");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "60% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Ace Pap 1 KG");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 3); //Checkers is the third store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*ace\_pap\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "F00DLOVER$R1");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "40% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Brown Rice 1 KG");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 4); //Food Lover's Market is the fourth store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*brownrice\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "F00DLOVER$UT");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "60% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "UTD Potatoes 1 KG");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 4); //Food Lover's Market is the fourth store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*potatoes\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "F00DLOVER$SC");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "70% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Straw-Choc Cake");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 4); //Food Lover's Market is the fourth store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*cake\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
  
 values.clear(); //Clear the ContentValues object so that it can be reused:  
 values.put(*COLUMN\_DISCOUNT\_CODE*, "F00DLOVER$BB");  
 values.put(*COLUMN\_DISCOUNT\_PERCENTAGE*, "80% OFF");  
 values.put(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*, "Bread Rolls x6");  
 values.put(*COLUMN\_DISCOUNT\_RETAILER\_ID*, 4); //Food Lover's Market is the fourth store that is loaded:  
 values.put(*COLUMN\_DISCOUNT\_IMAGE*, convertImageToByteArray(R.drawable.*rolls\_compressed*));  
 db.insert(*TABLE\_NAME\_DISCOUNTED\_PRODUCTS*, null, values);  
 }  
  
 @Override  
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){ //Upgrade the database schema if necessary:  
 db.execSQL("DROP TABLE IF EXISTS " + *TABLE\_NAME\_CUSTOMER*);  
 db.execSQL("DROP TABLE IF EXISTS " + *TABLE\_NAME\_RETAILER*);  
 db.execSQL("DROP TABLE IF EXISTS " + *TABLE\_NAME\_DISCOUNTED\_PRODUCTS*);  
 onCreate(db);  
 }  
  
 //Use a get method for the email column:  
 public String getColumnEmail(){  
 return *COLUMN\_CUSTOMER\_EMAIL*;  
 }  
  
 //Use a get method for the password column:  
 public String getColumnPassword(){  
 return *COLUMN\_CUSTOMER\_PASSWORD*;  
 }  
}

This code does the following:

* Defines and creates the database tables and inserts initial data.
* Additional functions are present as they are used throughout the mobile application.

### Data Access Adapters

#### Accessor Methods – Account

**Code Snippet:**

private void loadCustomerDetails(){  
 SharedPreferences sharedPreferences = PreferenceManager.*getDefaultSharedPreferences*(getApplicationContext());  
 String email = sharedPreferences.getString("email", "");  
  
 DatabaseHelper databaseHelper = new DatabaseHelper(this);  
 SQLiteDatabase db = databaseHelper.getReadableDatabase();  
  
 String[] projection = {  
 DatabaseHelper.*COLUMN\_CUSTOMER\_NAME*,  
 DatabaseHelper.*COLUMN\_CUSTOMER\_SURNAME*,  
 DatabaseHelper.*COLUMN\_CUSTOMER\_EMAIL* };  
  
 String selection = DatabaseHelper.*COLUMN\_CUSTOMER\_EMAIL* + " =?";  
 String[] selectionArgs = {email};  
  
 Cursor cursor = db.query(  
 DatabaseHelper.*TABLE\_NAME\_CUSTOMER*,  
 projection,  
 selection,  
 selectionArgs,  
 null,  
 null,  
 null  
 );  
  
 if(cursor != null && cursor.moveToFirst()){  
 int customerNameIndex = cursor.getColumnIndex(DatabaseHelper.*COLUMN\_CUSTOMER\_NAME*);  
 int customerSurnameIndex = cursor.getColumnIndex(DatabaseHelper.*COLUMN\_CUSTOMER\_SURNAME*);  
 int customerEmailIndex = cursor.getColumnIndex(DatabaseHelper.*COLUMN\_CUSTOMER\_EMAIL*);  
  
  
 String customerName = cursor.getString(customerNameIndex);  
 String customerSurname = cursor.getString(customerSurnameIndex);  
 String customerEmail = cursor.getString(customerEmailIndex);  
  
 cursor.close();  
 db.close();  
  
 nameTextView.setText("Name: " + customerName);  
 surnameTextView.setText("Surname: " + customerSurname);  
 emailTextView.setText("Email: " + customerEmail);  
 }  
}

*Figure 23 - Extract from the AccountActivity.java*

#### Accessor Methods – Products

//Function to fetch the discounted products in the discounted products table based on the Retailer ID:  
public List<DiscountedProduct> getDiscountedProductsByRetailerId(int retailerId) {  
 List<DiscountedProduct> discountedProducts = new ArrayList<>();  
 SQLiteDatabase database = getReadableDatabase();  
  
 String[] columns = {  
 *COLUMN\_DISCOUNT\_RETAILER\_ID*,  
 *COLUMN\_DISCOUNT\_CODE*,  
 *COLUMN\_DISCOUNT\_PERCENTAGE*,  
 *COLUMN\_DISCOUNT\_PRODUCT\_NAME*,  
 *COLUMN\_DISCOUNT\_IMAGE* };  
  
 String selection = *COLUMN\_DISCOUNT\_RETAILER\_ID* + " = ?";  
 String[] selectionArgs = { String.*valueOf*(retailerId) };  
  
 Cursor cursor = database.query(  
 *TABLE\_NAME\_DISCOUNTED\_PRODUCTS*,  
 columns,  
 selection,  
 selectionArgs,  
 null,  
 null,  
 null  
 );  
  
 int columnDiscountCode = cursor.getColumnIndex(*COLUMN\_DISCOUNT\_CODE*);  
 int columnDiscountPercentage = cursor.getColumnIndex(*COLUMN\_DISCOUNT\_PERCENTAGE*);  
 int columnProductName = cursor.getColumnIndex(*COLUMN\_DISCOUNT\_PRODUCT\_NAME*);  
 int columnDiscountImage = cursor.getColumnIndex(*COLUMN\_DISCOUNT\_IMAGE*);  
  
 if (cursor != null && cursor.moveToFirst()) {  
 do {  
 String discountCode = cursor.getString(columnDiscountCode);  
 String discountPercentage = cursor.getString(columnDiscountPercentage);  
 String productName = cursor.getString(columnProductName);  
 byte[] discountImage = cursor.getBlob(columnDiscountImage);  
  
 DiscountedProduct discountedProduct = new DiscountedProduct(discountCode, discountPercentage, productName, discountImage);  
 discountedProducts.add(discountedProduct);  
 } while (cursor.moveToNext());  
 }  
  
 cursor.close();  
 database.close();  
  
 return discountedProducts;  
}

## Conclusion

In summary, multiple methods were used when approaching the system prototyping and testing phase.

These methods include building on the initial designs to develop a visually appealing application, focusing on the core functionality of the application and using unit testing to ensure it is functioning as it should be, finding the most appropriate database implementation technology to better manage the user data, and combing these aspects to create a functioning mobile application.

Using iterative and incremental design, the application has been built on that has been conceptualized in deliverable one. While this remains an ongoing process, great strides were made in building on the initial concepts and move further towards a mobile application that meets all of the requirements.

# List of Figures

[*Figure 1 - Landing screen when the user opens the application for the first time.* 9](#_3fwokq0)

[*Figure 2 - Landing Page when the user opens the application for the first time – Emulator Pixel 6 Pro API 30* 9](#_1v1yuxt)

[*Figure 3 – Extract from MainActivity.java* 10](#_sqyw64)

[*Figure 4 - Registration screen so that users can create an account.* 11](#_4f1mdlm)

[*Figure 5 – Registration screen - Emulator Pixel 6 Pro API 30* 11](#_2u6wntf)

[*Figure 6 – Extract from RegisterActivity.java* 12](#_3cqmetx)

[*Figure 7 - Login Page so that users can login using the account they have just created.* 13](#_19c6y18)

[*Figure 8 – Login Screen - Emulator Pixel 6 Pro API 30* 13](#_3tbugp1)

[*Figure 9 – Extract from LoginActivity.java* 14](#_28h4qwu)

[*Figure 10 – Retail partners page for the Sustainable Spoonful Mobile Application.* 15](#_nmf14n)

[*Figure 11 – Home Screen - Emulator Pixel 6 Pro API 30* 15](#_37m2jsg)

[*Figure 12 – LandingActivity.java* 16](#_1rvwp1q)

[*Figure 13 - Discounted Products Page for users to search for discounted products based on the store and location.* 17](#_1mrcu09)

[*Figure 14 – Discounts Screen - Emulator Pixel 6 Pro API 30* 17](#_46r0co2)

[*Figure 15 – Extract from DiscountActivity.java* 19](#_2lwamvv)

[*Figure 16 - Discounts listed for a specific store.* 20](#_111kx3o)

[*Figure 17 - Products Screen - Emulator Pixel 6 Pro API 30* 20](#_3l18frh)

[*Figure 18 – User account screen* 21](#_206ipza)

[Figure 19 – User account Screen - *Emulator Pixel 6 Pro API 30* 22](#_4k668n3)

[*Figure 20 – Extract From AccountActivity.java* 23](#_4bvk7pj)

[*Figure 21 – Extract from LoginActivity.java* 24](#_2r0uhxc)

[*Figure 22 – Extract from the RegisterActivity.java* 26](#_1664s55)

[Figure 24 – Extract from DatabaseHelper.java 28](#_2zbgiuw)

[*Figure 23 - Extract from the AccountActivity.java* 29](#_1egqt2p)

# List of Tables

[Table 1: Roles and responsibilities 5](#_2grqrue)

[Table 2: Testing Schedule and Report 8](#_vx1227)

# Bibliography

AllCodingTutorials, 2022. *RecyclerView with SQLite Database | Populate RecyclerView with SQLite Database | Source Code*. [Video] Available at: https://youtu.be/6enCz2cY18c [Accessed 21 June 2023].

Android Coding, 2019. *How to Implement Bottom Navigation With Activities in Android Studio | BottomNav | Android Coding*. [Video] Available at: https://youtu.be/JjfSjMs0ImQ [Accessed 21 June 2023].

Android Developers, 2023. *Save data using SQLite*. [Online] Available at: https://developer.android.com/training/data-storage/sqlite#java [Accessed 21 June 2023].

BTech Days, 2021. *Login and Register Form using SQLite Database in Android Studio | Login Registration Android Studio*. [Video] Available at: https://youtu.be/o9CVZ1gQgQo [Accessed 21 June 2023].

Coding Adventure, 2020. *CardView UI Design Android Studio | Using Grid Layout*. [Video] Available at: https://youtu.be/YKssd\_9x8Eg [Accessed 21 June 2023].

Coding With Me, 2021. *LOGIN APP IN ANDROID STUDIO USING SQLite | SQLite DATABASE | LOGIN AND REGISTER APP WITH FULL CODING*. [Video] Available at: https://youtu.be/yJ02XTKiuAc [Accessed 11 March 2022].

CodingWithSaani, 2019. *2- Save and Retrieve Images with SQLite in Android Studio 2019 | Getting Image from Device*. [Video] Available at: https://youtu.be/9bUkyop6eMg [Accessed 21 June 2023].

CodingZest, 2021. *Modern Dashboard UI Design Android Studio Tutorial | Cardview Android Studio | Grid layout Android*. [Video] Available at: https://youtu.be/ixRXEoGAEZM [Accessed 21 June 2023].

Deccon Tech, 2022. *How to insert image and retrieve in sqlite database Android Studio.* [Video] Available at: https://youtu.be/wGBhi9Vf7yA [Accessed 21 June 2023].

Fowler, M., 2017. *Martin Fowler.* [Online] Available at: https://martinfowler.com/ [Accessed 16 June 2023].

GeeksForGeeks, 2019. *Software Testing | Functional Testing*. [Online] Available at: https://www.geeksforgeeks.org/software-testing-functional-testing/ [Accessed 21 June 2023].

Gillis, A., 2023. *What is User Acceptance Testing (UAT)?* [Online] Available at: https://www.techtarget.com/searchsoftwarequality/definition/user-acceptance-testing-UAT [Accessed 21 June 2023].

Hamilton, T., 2019. *Unit Testing Tutorial: What is, Types, Tools, EXAMPLE.* [Online] Available at: https://www.guru99.com/unit-testing-guide.html[Accessed 17 June 2023].

JavaTPoint, 2021. *Performance Testing - javatpoint*. [Online] Available at: https://www.javatpoint.com/performance-testing [Accessed 21 June 2023].

JavaTPoint, 2011. *Functional Testing - javatpoint*. [Online] Available at: https://www.javatpoint.com/functional-testing [Accessed 21 June 2023].

JRebel, 2023. *How to Use Java Integration Testing | JRebel & XRebel by Perforce*. [Online] Available at: https://www.jrebel.com/blog/how-to-use-java-integration-testing [Accessed 21 June 2023].

Lemon Soft, 2022. *How to Implement Bottom Navigation With Activities in Android Studio | BottomNav | Lemon Soft*. [Video] Available at: https://youtu.be/y4arA4hsok8 [Accessed 21 June 2023].

Nalawade, K., 2023. *How to Write Unit Tests in Java*. [Online] Available at: https://www.freecodecamp.org/news/java-unit-testing/#:~:text=What%20is%20Unit%20Testing%3F [Accessed 21 June 2023].

Ng, C., 2021. *What is data integrity and how can you maintain it?.* [Online] Available at: https://www.varonis.com/blog/data-integrity[Accessed 16 June 2023].

Optimizely, 2023. *Usability testing.* [Online] Available at: https://www.optimizely.com/optimization-glossary/usability-testing/#:~:text=The%20goal%20of%20usability%20testing[Accessed 21 June 2023].

pandaquests, 2023. *Acceptance testing in JavaScript*. [Online] Available at: https://levelup.gitconnected.com/acceptance-testing-in-javascript-ca1db5469fe9 [Accessed 21 June 2023].

Practical Coding, 2021. *RecyclerView | Everything You Need to Know*. [Video] Available at: https://youtu.be/Mc0XT58A1Z4 [Accessed 28 December 2022].

Quesenbery, 2023. *What Does Usability Mean: Looking Beyond ‘Ease of Use’ - Whitney Interactive Design.* [Online] Available at: https://www.wqusability.com/articles/more-than-ease-of-use.html#:~:text=An%20understanding%20of%20the%20five[Accessed 17 June 2023].

Sahani, B., 2018. *SQLiteBlobTooBigException: Row too big to fit into CursorWindow requiredPos=0, totalRows=1*. [Online] Available at: https://stackoverflow.com/questions/51959944/sqliteblobtoobigexception-row-too-big-to-fit-into-cursorwindow-requiredpos-0-t [Accessed 21 June 2023].

Soegaard, M., 2019. *Usability: A part of the User Experience.* [Online]   
Available at: https://www.interaction-design.org/literature/article/usability-a-part-of-the-user-experience[Accessed 16 June 2023].

Software Testing Help, 2014. *How to Write an Effective Test Summary Report [Download Template]*. [Online] Available at: https://www.softwaretestinghelp.com/test-summary-report-template-download-sample/ [Accessed 21 June 2023].

SQLite, 2023. *SQLite.* [Online] Available at: https://www.sqlite.org/index.html  
[Accessed 19 June 2023].

Usability.gov, 2019. *Planning a Usability Test | Usability.gov.* [Online]   
Available at: https://www.usability.gov/how-to-and-tools/methods/planning-usability-testing.html  
[Accessed 18 June 2023].

W3Schools, 2023. *XML Comments*. [Online] Available at: https://www.w3schools.in/xml/comments [Accessed 21 June 2023].

Wikipedia Contributors, 2023. *Unit\_testing*. [Online] Unit Testing. Available at: https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Unit\_testing&id=1156746873&wpFormIdentifier=titleform [Accessed 21 June 2023].

Zappa, L., 2015. *What is usability testing | checklist | methods | tool and cost | examples and cases*. [Online] Available at: https://www.userreport.com/blog/usability-testing/ [Accessed 21 June 2023].

Zinchenko, 2019. *Usability testing: the road to better conversions and user retention – MindK Blog.* [Online] Available at: https://www.mindk.com/blog/usability-testing/[Accessed 17 June 2023].