**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 3**](#_gjdgxs)

[4.1](#_30j0zll) Introduction 3

[4.2](#_1fob9te) Testing Plan 5

[4.3](#_3znysh7) Layouts Development 9

[*4.3.1*](#_2et92p0) *User Interface – Main Screen 9*

[4.3.1.1.1](#_tyjcwt) Iteration 1 9

[4.3.1.1.2](#_3dy6vkm) Iteration 2 10

[*4.3.2*](#_1t3h5sf) *User Interface - Registration 12*

[4.3.2.1.1](#_4d34og8) Iteration 1 12

[4.3.2.1.2](#_2s8eyo1) Iteration 2 12

[*4.3.3*](#_17dp8vu) *User Interface - Login 16*

[4.3.3.1.1](#_3rdcrjn) Iteration 1 16

[4.3.3.1.2](#_26in1rg) Iteration 2 16

[*4.3.4*](#_lnxbz9) *User Interface – Retail Partners 19*

[4.3.4.1.1](#_35nkun2) Iteration 1 19

[4.3.4.1.2](#_1ksv4uv) Iteration 2 20

[*4.3.5*](#_44sinio) *User Interface – Discounts 22*

[4.3.5.1.1](#_2jxsxqh) Iteration 1 22

[4.3.5.1.2](#_z337ya) Iteration 2 22

[*4.3.6*](#_3j2qqm3) *User Interface – Products 25*

[4.3.6.1.1](#_1y810tw) Iteration 1 25

[4.3.6.1.2](#_4i7ojhp) Iteration 2 25

[*4.3.7*](#_2xcytpi) *User Interface – Account 26*

[4.3.7.1.1](#_1ci93xb) Iteration 1 26

[4.3.7.1.2](#_3whwml4) Iteration 2 27

[4.4](#_2bn6wsx) Business Logic Development 31

[*4.4.1*](#_qsh70q) *Algorithm 1 – Login 31*

[*4.4.2*](#_3as4poj) *Algorithm 2 - Registration 33*

[4.5](#_1pxezwc) Data Access Development 36

[*4.5.1*](#_49x2ik5) *Database Implementation 36*

[4.5.1.1](#_2p2csry) Database Creation Class – DatabaseHelper.java 36

[*4.5.2*](#_147n2zr) *Data Access Adapters 38*

[4.5.2.1](#_3o7alnk) Accessor Methods – Account 38

[4.5.2.2](#_23ckvvd) Accessor Methods – Products 39

[4.6](#_ihv636) Conclusion 39

[**List of Figures 40**](#_32hioqz)

[**List of Tables 40**](#_1hmsyys)

[**References 41**](#_41mghml)

# 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, 2021). The mobile application should be easy to navigate through and be responsive on mobile devices (Soegaard, 2021). The design of the screens should remain consistent throughout the mobile application (Soegaard, 2021). Usability testing should be conducted to ensure that the user can navigate and interact with the mobile application easily (Soegaard, 2021). This can help to ensure that users will have a positive user experience when using this mobile application (Soegaard, 2021).

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 (Buckbee, 2021). During this stage, it is important that the integrity of the user and mobile application’s data is maintained and secured (Buckbee, 2021). It is also imperative that an efficient and scalable database is developed, this can assist in minimizing response times by optimizing data retrieval (Buckbee, 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, 2023). By taking these steps, this can ensure that a stable and reliable mobile application will be developed (Hamilton, 2023).

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 |
| Santana | * Proofreader | * Proofread the deliverable 2 document. |

**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, 2023).
* The screens in this mobile application are (Usability.gov, 2023):
  + 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 application lagging after registration.

Recommendations:

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

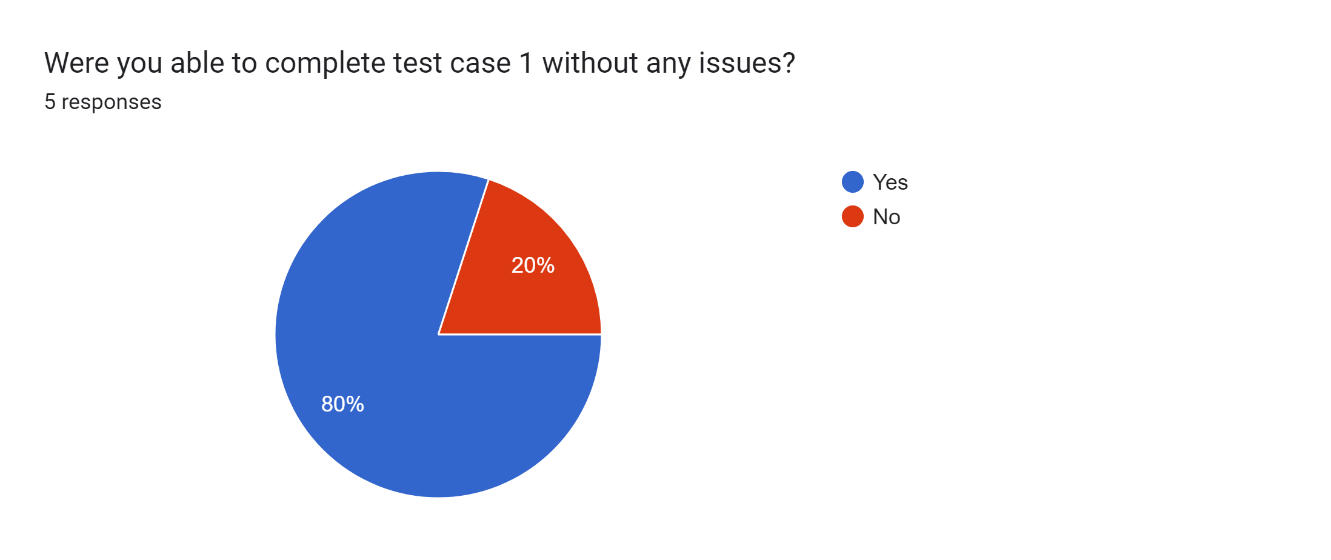


Figure 1 – Test Case 1 - Results Graph

**Test Case 2**

Analysis:

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

Recommendations:

* Keep building on current iteration.

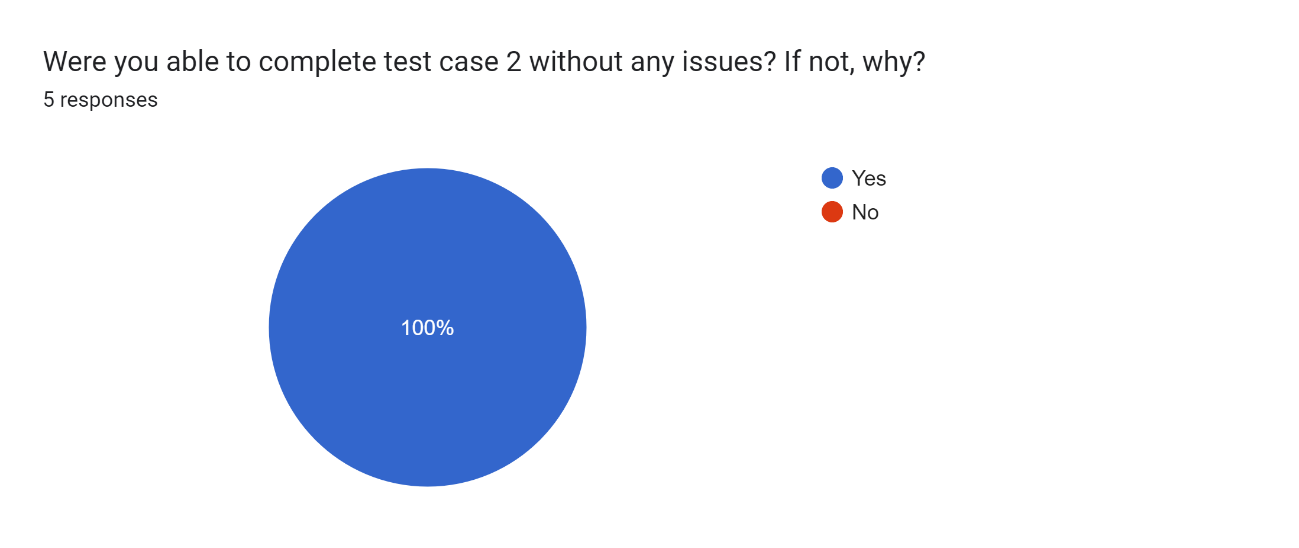
****

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

Recommendations:

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

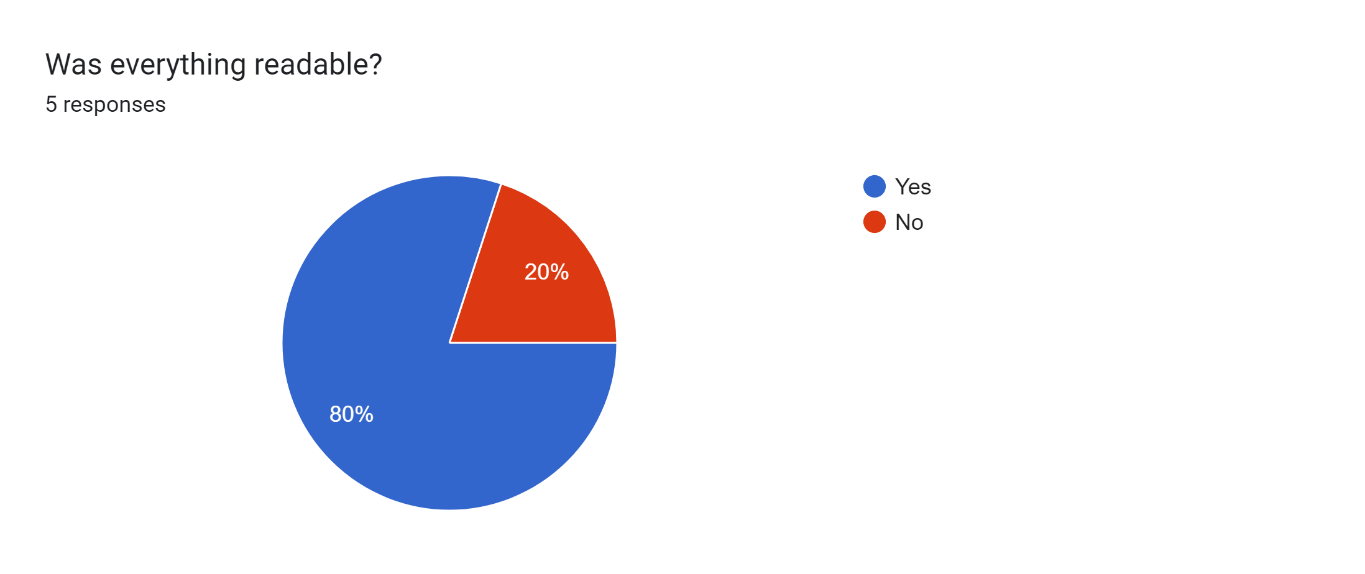


Figure 6 – Was everything readable - Result Graph

Analysis:

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

Recommendations:

* Keep building on current iteration.

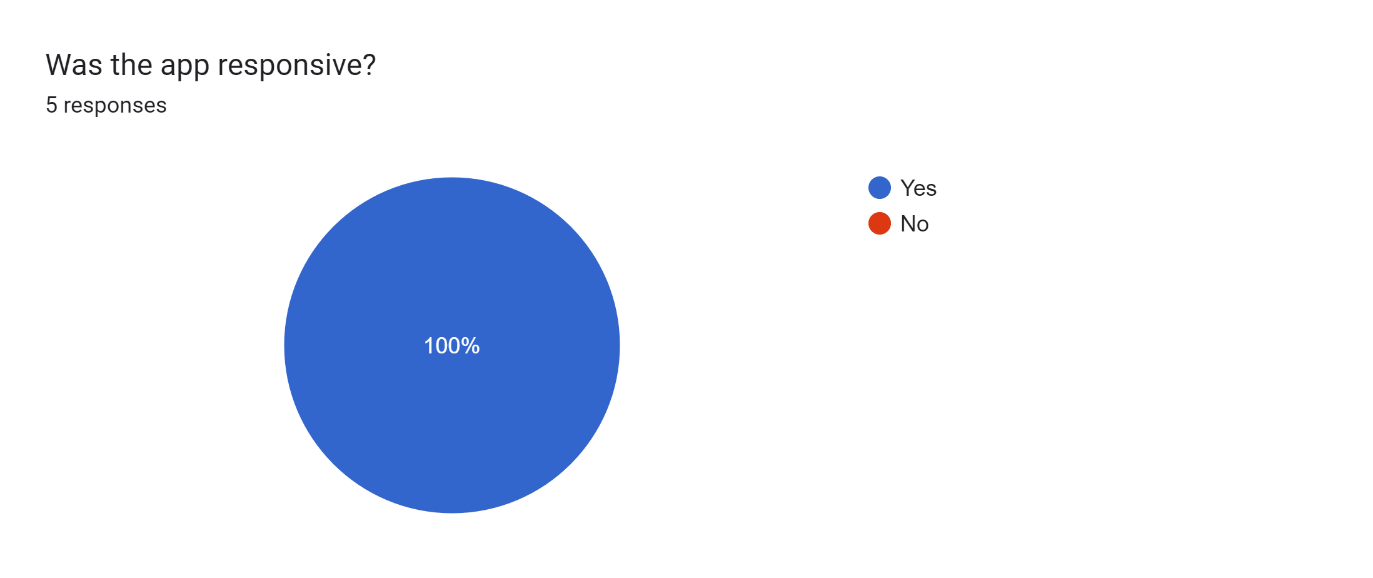


Figure 7 –Was the application responsive - Result Graph

Analysis:

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

Recommendations:

* Keep building on current iteration.

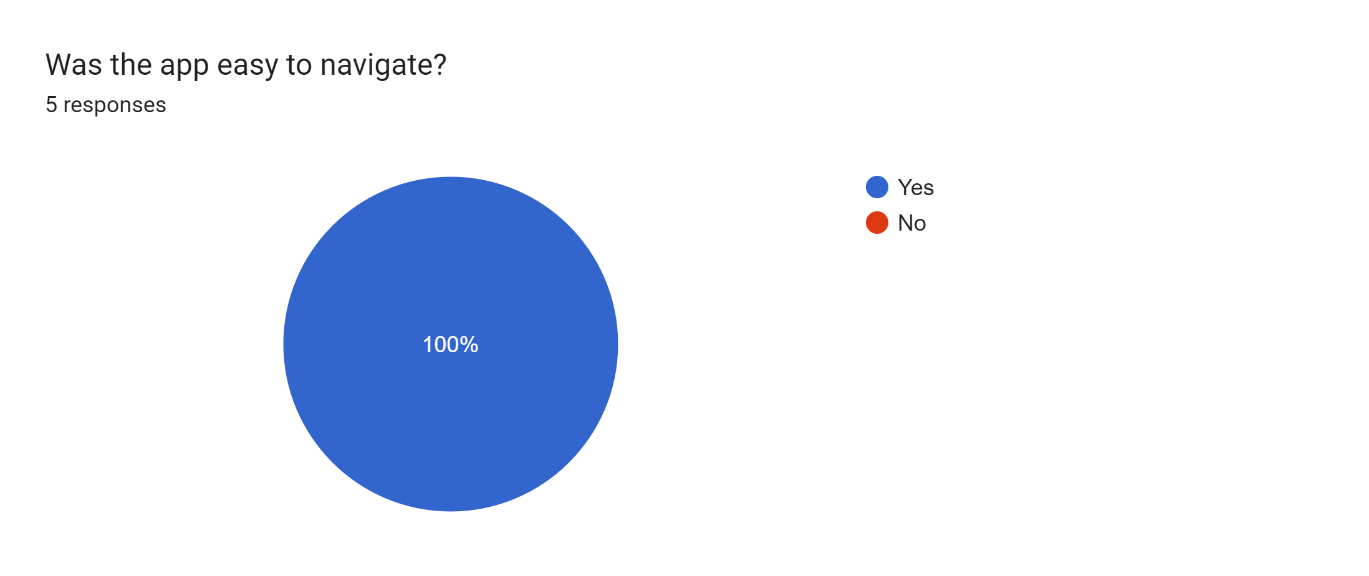


Figure 8 – Was the application easy to navigate – Result Graph

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 initial screen the user will be presented with when open the application on their mobile device. Users are presented with a simple, user-friendly interface where they can either Register for an account or login with their existing account.

Should the user already be logged in, the user will automatically be redirected to the home page removing the need for repetitive login procedures.

By offering both the login and registration options on the initial screen, both types of users will be catered to ensuring a personalized experience for all.

##### 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:**

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:  
  
 //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);  
 }  
 });  
 }  
}

This snippet for the Main Screen does the following:

* Creates the main screen when opening the mobile application.
* From here the user can register for an account or login.
* If the user is already logged in, it will redirect them to the Main/ Home Screen.

### User Interface – Registration

The registration screen 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 database and the user will be redirected to the login screen. An alert message will be displayed notifying the user that their registration was successful.

##### 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:**

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 for the Registration Screen does the following:

* Creates the registration screen.
* Creates an instance of the DatabaseHelper.
* Inserts the user’s details into the database.

### User Interface – Login

The login screen allows the user to enter their login credentials. The 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 database. Once verified the user will be redirected to the landing page. An alert (toast) will be displayed notifying the user that they have been successfully logged in.

##### Iteration 1

A screen shot of a phone

Description automatically generated with medium confidence

*Figure 7 - Login Page so that users can login using the account they have just created.*

##### Iteration 2

A screen shot of a cell phone

Description automatically generated with low confidence

*Figure 8 – Login Screen - Emulator Pixel 6 Pro API 30*

**Code Snippet:**

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 for the Login Screen does the following:

* Creates the login screen.
* Created an instance of the DatabaseHelper class
* Setups a listener on the login button.

### User Interface – Retail Partners

##### Iteration 1

A screen shot of a phone

Description automatically generated with medium confidence

*Figure 10 – Retail partners page for the Sustainable Spoonful Mobile Application.*

##### Iteration 2

A screen shot of a cell phone

Description automatically generated with medium confidence

*Figure 11 – Home Screen - Emulator Pixel 6 Pro API 30*

This is the home Screen. When the user successfully logs in, they will be redirected to this screen. From here the user can navigate using the navbar at the bottom of the screen.

**Code Snippet:**

public class LandingActivity extends AppCompatActivity {  
  
 BottomNavigationView bottom\_nav\_bar;  
  
 @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;  
 }  
 }  
 });  
 }  
  
 @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:  
 }  
 }  
}

This snippet for the Home Screen does the following:

* Creates the home screen.
* Setups up the navbar.
  + 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.

### User Interface – Discounts

This is the discount screen. By pressing the search icon at the bottom of the screen the user will be directed to this screen where they can see a summary of available discounts arranged by store. From here they can select a store by pressing on the store card. They will then be redirected to the discounted products screen.

##### Iteration 1

A cell phone with text on the screen

Description automatically generated with low confidence

*Figure 13 - Discounted Products Page 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 - Emulator Pixel 6 Pro API 30*

Code Snippet:

protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_discount*);  
  
 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;  
 }  
 }  
 });  
  
 //For the cards on the layout screen  
 cardPicknpay = findViewById(R.id.*cardPicknpay*);  
 cardCheckers = findViewById(R.id.*cardCheckers*);  
 cardWoolworths = findViewById(R.id.*cardWoolworths*);  
 cardFoodLoversMarket = findViewById(R.id.*cardFoodLoversMarket*);  
  
 cardPicknpay.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 Intent intent = new Intent(DiscountActivity.this,PicknPayProductsActivity.class);  
 startActivity(intent);  
 }  
 });  
 cardCheckers .setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 Intent intent = new Intent(DiscountActivity.this,CheckersProductsActivity.class);  
 startActivity(intent);  
 }  
 });  
 cardWoolworths .setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 Intent intent = new Intent(DiscountActivity.this,WoolworthsProductsActivity.class);  
 startActivity(intent);  
 }  
 });  
 cardFoodLoversMarket.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 Intent intent = new Intent(DiscountActivity.this,FoodLoversProductsActivity.class);  
 startActivity(intent);  
 }  
 });  
}

*Figure 15 – Extract from DiscountActivity.java*

This snippet for the Discounts Screen does the following:

* Sets the search icon to be displayed when 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.

### User Interface – Products

##### Iteration 1

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

Description automatically generated

*Figure 16 - Discounts listed for a specific store.*

##### Iteration 2

A screen shot of a phone

Description automatically generated with low confidence

*Figure 17 - Products Screen - Emulator Pixel 6 Pro API 30*

This is the products screen. 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 and explanation

This snippet for the Products Screen does the following:

### User Interface – Account

##### Iteration 1

A screen shot of a cell phone

Description automatically generated with medium confidence

*Figure 18 – User account screen*

##### Iteration 2

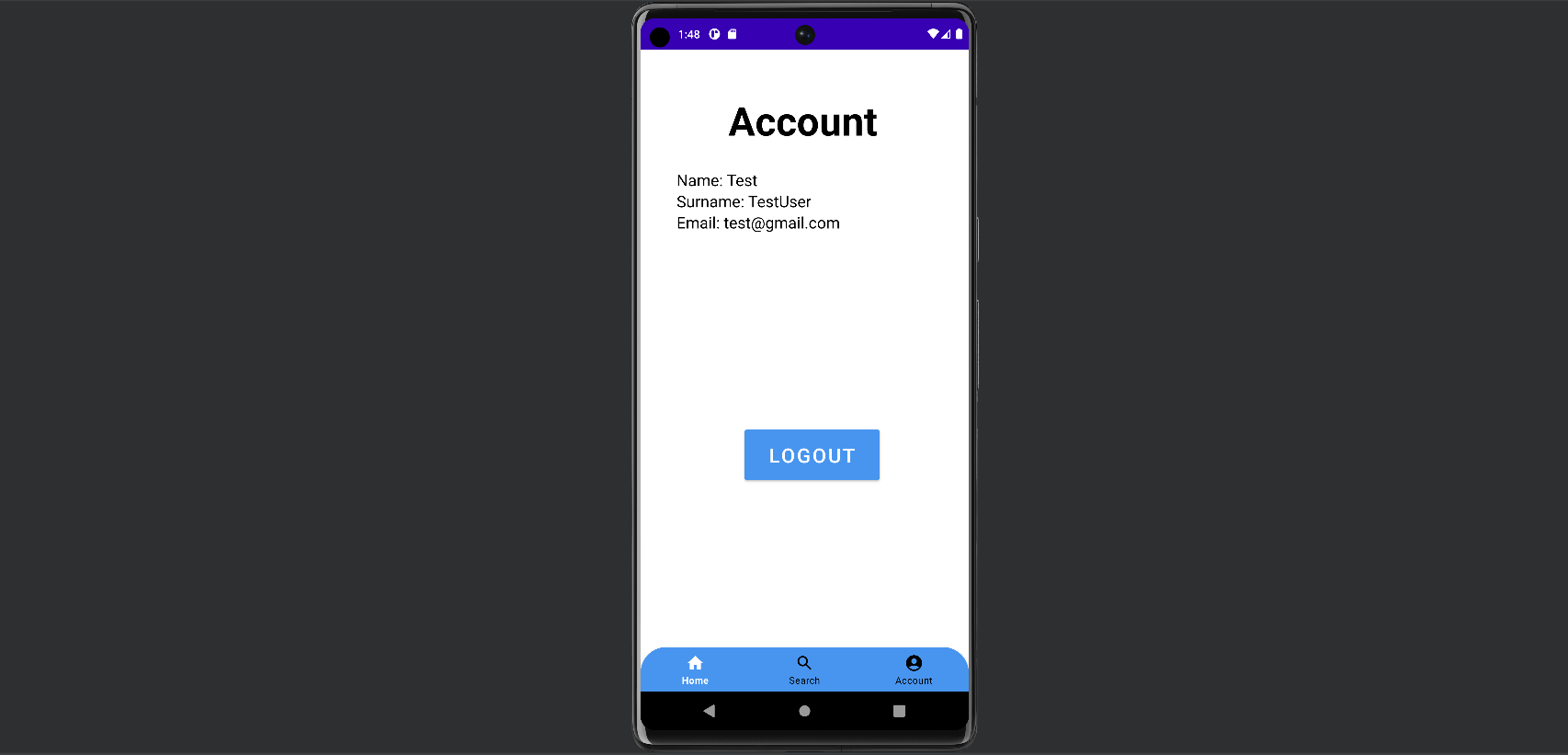


Figure 19 – User account Screen - *Emulator Pixel 6 Pro API 30*

This is the user accounts screen. The 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 snippet for the Products Screen does the following:

* Creates the account screen.
* Checks if the customer is logged, if not a message will be displayed, and they will be redirected to the Main Screen (Login or Register).
* Sets the account icon to be selected when on this screen.
* Loads the loadCustomerDetails() function.

## 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.content.Context;  
import android.database.sqlite.SQLiteDatabase;  
import android.database.sqlite.SQLiteOpenHelper;  
  
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";  
  
 //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 DatabaseHelper(Context context){  
 super(context, *DATABASE\_NAME*, null, *DATABASE\_VERSION*);  
 }  
  
 @Override  
 public void onCreate(SQLiteDatabase db){  
 //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)";  
 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," +  
 "FOREIGN KEY (" + *COLUMN\_DISCOUNT\_RETAILER\_ID* + ") REFERENCES " +  
 *TABLE\_NAME\_RETAILER* + "(" + *COLUMN\_RETAILER\_ID* + "))";  
 db.execSQL(createDiscountedProductsTable);  
 }  
  
 @Override  
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){  
 //Upgrade the database 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*;  
 }  
}

*Figure 24 – Extract from DatabaseHelper.java*

This snippet for the Discounts Screen does the following:

### 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*

This snippet of code for registration does the following:

#### Accessor Methods – Products

## 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)

# References

Anon, 2023. *How To Write An Effective Test Summary Report.* [Online]   
Available at: https://www.softwaretestinghelp.com/test-summary-report-template-download-sample/

Buckbee, M., 2021. *What is Data Integrity and How Can You Maintain it?.* [Online]   
Available at: https://www.varonis.com/blog/data-integrity  
[Accessed 17 June 2023].

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

Fowler, M., 2017. *Martin Fowler.* [Online]   
Available at: https://martinfowler.com/  
[Accessed 17 June 2023].

Gillis, A. S., 2023. *user acceptance testing (UAT).* [Online]   
Available at: https://www.techtarget.com/searchsoftwarequality/definition/user-acceptance-testing-UAT

Hamilton, T., 2023. Integration Testing: What is, Types with Example. *GURU99,* p. 5.

Hamilton, T., 2023. Performance Testing Tutorial – Types. *GURU99,* p. 5.

Hamilton, T., 2023. TEST PLAN: What is, How to Create. *GURU99,* pp. 3-4.

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

JavaTPoint, 2021. Functional Testing. *JavaTPoint,* p. 12.

JavaTpoint, 2023. Performance Testing. *JavaTpoint,* p. 2.

JRebel, 2023. *How to Use Java Integration Testing.* [Online]   
Available at: https://www.jrebel.com/blog/how-to-use-java-integration-testing

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,as%20expected%20before%20releasing%20it.

Optimizely, 2023. *Usability testing.* [Online]   
Available at: https://www.optimizely.com/optimization-glossary/usability-testing/#:~:text=The%20goal%20of%20usability%20testing,go%20to%20a%20competitor's%20site.  
[Accessed 16 June 2023].

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

pp\_pankaj, 2023. *Software Testing – Functional Testing.* [Online]   
Available at: https://www.geeksforgeeks.org/software-testing-functional-testing/

Quesenbery, W., 2023. *What Does Usability Mean: Looking Beyond ‘Ease of Use’.* [Online]   
Available at: https://www.wqusability.com/articles/more-than-ease-of-use.html#:~:text=An%20understanding%20of%20the%20five,the%20goal%20of%20usable%20products.  
[Accessed 16 June 2023].

RichardJohnsonbaugh, 2009. *Discrete Mathematics.* United states of America: Pearson Education.

Soegaard, M., 2021. *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 17 June 2023].

SQLite, 2023. *SQLite.* [Online]   
Available at: https://www.sqlite.org/index.html  
[Accessed 16 June 2023].

Usability.gov, 2023. *Planning a Usability Test.* [Online]   
Available at: https://www.usability.gov/how-to-and-tools/methods/planning-usability-testing.html  
[Accessed 16 June 2023].

Zinchenko, P., 2019. *Usability testing: how to analyze your website for UX pitfalls.* [Online]   
Available at: https://www.mindk.com/blog/usability-testing/  
[Accessed 16 June 2023].