**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

Supervisors:

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](#_Toc138019634)

[4.1 Introduction 4](#_Toc138019635)

[4.2 Testing Plan 6](#_Toc138019636)

[4.3 Layouts Development 9](#_Toc138019637)

[4.3.1 User Interface – Main Screen 9](#_Toc138019638)

[4.3.1.1.1 Iteration 1 9](#_Toc138019639)

[4.3.1.1.2 Iteration 2 9](#_Toc138019640)

[4.3.2 User Interface - Registration 11](#_Toc138019641)

[4.3.2.1.1 Iteration 1 11](#_Toc138019642)

[4.3.2.1.2 Iteration 2 11](#_Toc138019643)

[4.3.3 User Interface - Login 13](#_Toc138019644)

[4.3.3.1.1 Iteration 1 13](#_Toc138019645)

[4.3.3.1.2 Iteration 2 13](#_Toc138019646)

[4.3.4 User Interface – Retail Partners 15](#_Toc138019647)

[4.3.4.1.1 Iteration 1 15](#_Toc138019648)

[4.3.4.1.2 Iteration 2 15](#_Toc138019649)

[4.3.5 User Interface – Discounts 17](#_Toc138019650)

[4.3.5.1.1 Iteration 1 17](#_Toc138019651)

[4.3.5.1.2 Iteration 2 17](#_Toc138019652)

[4.3.6 User Interface – Products 20](#_Toc138019653)

[4.3.6.1.1 Iteration 1 20](#_Toc138019654)

[4.3.6.1.2 Iteration 2 20](#_Toc138019655)

[4.4 Business Logic Development 21](#_Toc138019656)

[4.4.1 Algorithm 1 – Login 21](#_Toc138019657)

[4.4.2 Algorithm 2 - Registration 22](#_Toc138019658)

[4.5 Data Access Development 24](#_Toc138019659)

[4.5.1 Database Implementation 24](#_Toc138019660)

[4.5.1.1 Database Creation Class – DatabaseHelper.java 24](#_Toc138019661)

[4.5.2 Data Access Adapters 26](#_Toc138019662)

[4.5.2.1 Accessor Methods – Account 26](#_Toc138019663)

[4.6 Conclusion 26](#_Toc138019664)

[List of Figures 27](#_Toc138019665)

[List of Tables 27](#_Toc138019666)

[References 28](#_Toc138019667)

# System Prototype Development and Testing

## Introduction

Our goal with The Sustainable Spoonful has been to develop a mobile application assist in mitigating food wastage amongst consumers and retailers. Below we will attempt to provide some additional information regarding our implementation plan and our overall strategy when approaching this applications development.

When approaching the system prototype development and testing aspect of The Sustainable Spoonful mobile application we needed to consider the process we would follow to ensure we created a visually appealing, user-friendly mobile application that met our functional requirements. This would involve several stages.

These stages include layouts development, business logic development, data access development and testing.

The layouts development stage was where we attempted to create recreate and build upon our prototype from deliverable 3, which as it stands consists of a user registration screen, a login screen, store listing screen, discounted product screen and the discount code screen. In this stage we need to ensure we are prioritizing the user experience by creating something that is both easy to navigate and responsive while remaining consistent as well as conducting usability testing based on user interactions and experiences. (Soegaard, 2015)

The business logic development stage was where we attempted to translate our business requirements into functional code. During this stage we need to ensure we have clearly defined our business requirements so that can translate it into functional code as well as attempting to follow best practices, ensuring maintainability, readability, and scalability. We can achieve this by adopting an agile approach and adjusting our code throughout the process. (Fowler, 2017)

The data access development stage was where we designed our data access layer that will interact with our applications backend systems and the SQLite database, and finally testing where we meticulously tested our application to ensure its quality and reliability. During this stage we also need to ensure that we maintain data integrity as well as ensuring that user data remains secure. We also need to design an efficient as well as scalable database to minimize response times by optimizing data retrieval. (Buckbee, 2021)

Finally, with regards to testing, we need to establish a testing strategy and create test cases, ensuring we test all features and functionalities of the application as well as documenting reported issues to address them promptly to maintain a stable and reliable application. (Hamilton, 2023)

During these various stages, we have allocated different responsibilities to different team members to ensure clarity and responsibility throughout the planning and development process.

|  |  |  |
| --- | --- | --- |
| **Roles and Responsibilities** | | |
| **Name** | **Role** | **Responsibilities** |
| Melany | Frontend Developer and proofreader | Focus on designing and developing the visual aspects of the application.  Proofread deliverable 4 document |
| Lea | Backend Developer and proofreader | Focus on developing the functional aspects of the application  Proofread deliverable 4 document |
| Lucinda | Documentation and testing | Compile project documentation |
| Santana |  |  |

Table 1: Roles and responsibilities

Our goal with this project is to apply our research towards building an application that can make a difference.

## Testing Plan

During the stage of this deliverable, we will be evaluating whether the average user would the application easy to navigate. In this case we have chosen to focus on usability testing. We have created a series of test cases for our users to work through, once done we have setup a survey for them to answer and give us a bit of insight into their experience.

Objective:

* To understand how real users will interact with our application and make changes based on those results. (Optimizely, 2023)
* Ensure the application is effective, efficient, engaging, error tolerant and easy to learn.
  + **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 application with surveys or interviews.
  + **Error tolerant**: Create test case scenarios with the potential to cause errors.
  + **Easy to learn**: Try to get test users from different technical knowledge backgrounds to determine how user friendly the application is.

(Quesenbery, 2023)

Scope:

* The test will include the application in its entirety.
  + Main screen
  + Login screen
  + Registration screen
  + Store screen
  + Discounts screen
  + Account Screen
* This includes content and navigation.

(Usability.gov, 2023)

Components:

1. Set up a usability test consisting of various tasks the user would need to accomplish. These tasks will include:
   * Creating an account
   * Logging in to their account
   * Browsing the available discounts
   * View Account
   * Logout
2. Setup up a survey for users to complete after they have tested the mobile application.

(Optimizely, 2023)

Equipment:

* Testing will be conducted via the user’s mobile phones.

Test Cases:

* **Test Case 1**: The user installs the application and opens it for the first time. They need to register for an account and those credentials to login to the app.
  + In this scenario, the user would need to open the app, register an account, login and then locate the discounted products screen.
  + For the test case to be considered successful, the user needs to successfully register an again, and successfully login.
* **Test Case 2**: The user closes the app after viewing the discounted products screen. If they reopen the application, will they need to re-enter their login details?
  + In this scenario, the user would need to close the app from the open apps list on the phone and then open The Sustainable food app again.
  + For this test case to be considered successful, the user needs to open the app without needing to enter their login details 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 login screen?
  + In this scenario, the user would need to navigate to the Account screen, view their information and then press the logout button.
  + For this test case to be considered successful, the user would need to successfully navigate to Account screen, logout successfully and then be redirected to the login screen.
* **Test 4**: The user is asked to look for a discount 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 this test case to be considered successful, the user would need to navigate to the discounted products section without asking for assistance and complete the action in a reasonable amount of time.
* **Test 5**: The user is asked to navigate to the accounts screen and then navigate back to the discounted products page using the menu option. Were they able to easily navigate it?
  + In this scenario, the user would need to navigate to their accounts screen and then navigate back towards the discounted products screen.
  + For this test case to be considered successful, the user would need to navigate to the accounts screen and then to discounted products screen without asking for assistance and complete the action in a reasonable amount of time.

User Survey Questions:

We created a google form for users to complete once they had worked through the test cases.

Link to survey form: <https://docs.google.com/forms/d/e/1FAIpQLSc3EnTE__Z76rWmMWVvez45fT4MDG6OvFQgRelu5jnNebCrZg/viewform>

Test Report:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

Table 2: Testing Schedule and Report

(Zinchenko, 2019)

Analysis and recommendations:

## Layouts Development

### User Interface – Main Screen

##### Iteration 1

A screenshot of a blue screen

Description automatically generated with low confidence

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

##### Iteration 2

A screen shot of a cell phone

Description automatically generated with medium confidence

*Figure 2 - Landing Page when the user opens the application for the first time – Emulator Pixel 6 Pro API 30*

This is the screen the user will see when they open the application. From here they can either register for an account or login with their login details. If the user is already logged in, they will be directed to the landing page.

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);  
 }  
 });  
 }  
}

*Figure 3 – Extract from MainActivity.java*

This snippet for the Main Screen does the following:

* Creates the main screen when opening the 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

##### Iteration 1

A screen shot of a phone

Description automatically generated with low confidence

*Figure 4 - Registration screen so that users can create an account.*

##### Iteration 2

A screen shot of a cell phone

Description automatically generated with low confidence

*Figure 5 – Registration screen - Emulator Pixel 6 Pro API 30*

The registration screen allows 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 (toast) will be displayed notifying the user that registration was successful.

Code Snippet:

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();  
 }  
 });  
}

*Figure 6 – Extract from RegisterActivity.java*

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

##### 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*

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.

Code Snippet:

@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();  
 }  
 });  
}

*Figure 9 – Extract from LoginActivity.java*

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:

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;  
 }  
 }  
 });  
}

*Figure 12 – LandingActivity.java*

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

##### 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*

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.

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

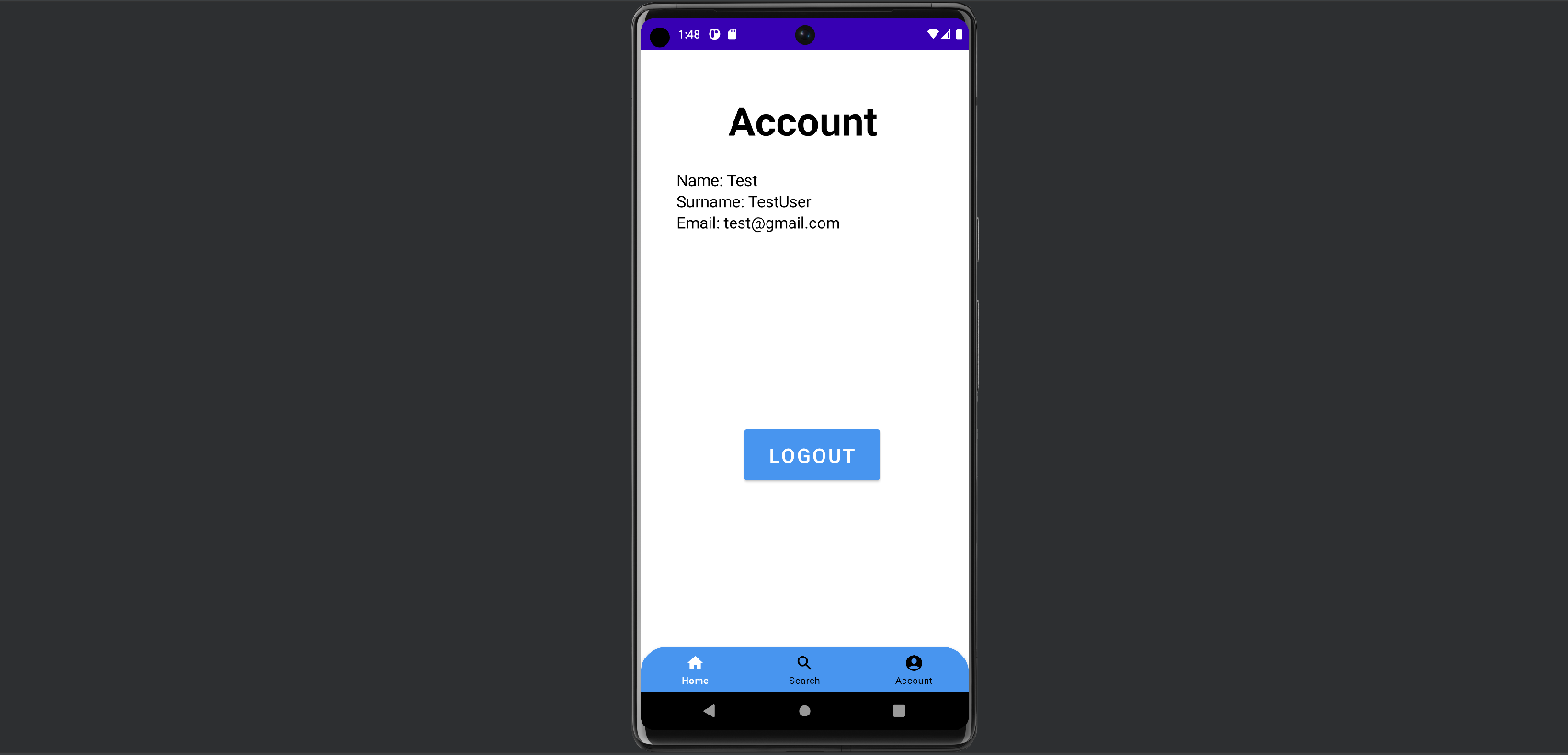


Figure 19 – User account Screen - *Emulator Pixel 6 Pro API 30*

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:

@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();  
}

*Figure 20 – Extract From AccountActivity.java*

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:

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();  
}

*Figure 21 – Extract from LoginActivity.java*

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:

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();  
 }  
}

*Figure 22 – Extract from the RegisterActivity.java*

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, we included it in our gradle.build file and created a new class called DatabaseHelper. We then created our tables and provided options for it to be created in other classes when the on create function is called. If it has been upgraded since its last use, we drop the table and recreate it again. We did this to 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, we have approached the system prototyping and testing phase using multiple methods.

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

Using iterative and incremental design we have managed to build on our application first conceptualized in deliverable one. While this remains an ongoing process, we have made great strides in building on our initial concepts and move further towards a mobile application that meets all our requirements.

# List of Figures

[*Figure 1 - Landing screen when the user opens the application for the first time.* 9](#_Toc138033593)

[*Figure 2 - Landing Page when the user opens the application for the first time – Emulator Pixel 6 Pro API 30* 9](#_Toc138033594)

[*Figure 3 – Extract from MainActivity.java* 10](#_Toc138033595)

[*Figure 4 - Registration screen so that users can create an account.* 11](#_Toc138033596)

[*Figure 5 – Registration screen - Emulator Pixel 6 Pro API 30* 11](#_Toc138033597)

[*Figure 6 – Extract from RegisterActivity.java* 12](#_Toc138033598)

[*Figure 7 - Login Page so that users can login using the account they have just created.* 13](#_Toc138033599)

[*Figure 8 – Login Screen - Emulator Pixel 6 Pro API 30* 13](#_Toc138033600)

[*Figure 9 – Extract from LoginActivity.java* 14](#_Toc138033601)

[*Figure 10 – Retail partners page for the Sustainable Spoonful Mobile Application.* 15](#_Toc138033602)

[*Figure 11 – Home Screen - Emulator Pixel 6 Pro API 30* 15](#_Toc138033603)

[*Figure 12 – LandingActivity.java* 16](#_Toc138033604)

[*Figure 13 - Discounted Products Page for users to search for discounted products based on the store and location.* 17](#_Toc138033605)

[*Figure 14 – Discounts Screen - Emulator Pixel 6 Pro API 30* 17](#_Toc138033606)

[*Figure 15 – Extract from DiscountActivity.java* 19](#_Toc138033607)

[*Figure 16 - Discounts listed for a specific store.* 20](#_Toc138033608)

[*Figure 17 - Products Screen - Emulator Pixel 6 Pro API 30* 20](#_Toc138033609)

[*Figure 18 – User account screen* 21](#_Toc138033610)

[Figure 19 – User account Screen - *Emulator Pixel 6 Pro API 30* 22](#_Toc138033611)

[*Figure 20 – Extract From AccountActivity.java* 23](#_Toc138033612)

[*Figure 21 – Extract from LoginActivity.java* 24](#_Toc138033613)

[*Figure 22 – Extract from the RegisterActivity.java* 26](#_Toc138033614)

[Figure 24 – Extract from DatabaseHelper.java 28](#_Toc138033615)

[*Figure 23 - Extract from the AccountActivity.java* 29](#_Toc138033616)

# List of Tables

[Table 1: Roles and responsibilities 5](#_Toc138021313)

[Table 2: Testing Schedule and Report 8](#_Toc138021314)

# 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. *Varonis.com.* [Online]   
Available at: https://www.varonis.com/blog/data-integrity  
[Accessed 17 June 2023].

Clifford Stein , Robert Drysdale, Kenneth Bogart, 2011. *Discrete Mathematics for computer scientists.* United states of America: Pearson Education.

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].

Gaddis, T., 2013. *Starting out with Programming Logic & Design.* United states: Pearson Education.

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

Hagarty, R., 2002. *Discrete Mathematics for computing.* Malaysia: Pearson Education limited.

Hamilton, T., 2023. *Guru99.* [Online]   
Available at: https://www.guru99.com/unit-testing-guide.html  
[Accessed 17 June 2023].

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.

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

Kolman, B., 1997. *Introductou Linear Algebra with applications.* New Yersey: Prentice -Hall, Inc.

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. *Optimizely.* [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. *WQUsability.* [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., 2015. *Interaction Design.* [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. *Usability.gov.* [Online]   
Available at: https://www.usability.gov/how-to-and-tools/methods/planning-usability-testing.html  
[Accessed 16 June 2023].

Zinchenko, P., 2019. *MindK.* [Online]   
Available at: https://www.mindk.com/blog/usability-testing/  
[Accessed 16 June 2023].