**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 |
| Bradbury, Santana | CNZJB3199 | 071 864 4170 | cnzjb3199@vossie.net |
| 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 |

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](#_Toc137856647)

[4.1 Introduction 3](#_Toc137856648)

[4.2 Testing Plan 5](#_Toc137856649)

[4.3 Layouts Development 8](#_Toc137856650)

[4.3.1 User Interface – Landing Page 8](#_Toc137856651)

[4.3.1.1.1 Iteration 1 8](#_Toc137856652)

[4.3.1.1.2 Iteration 2 9](#_Toc137856653)

[4.3.2 User Interface - Registration 10](#_Toc137856654)

[4.3.2.1.1 Iteration 1 10](#_Toc137856655)

[4.3.2.1.2 Iteration 2 10](#_Toc137856656)

[4.3.3 User Interface - Login 11](#_Toc137856657)

[4.3.3.1.1 Iteration 1 11](#_Toc137856658)

[4.3.3.1.2 Iteration 2 11](#_Toc137856659)

[4.3.4 User Interface - Stores 12](#_Toc137856660)

[4.3.4.1.1 Iteration 1 12](#_Toc137856661)

[4.3.4.1.2 Iteration 2 12](#_Toc137856662)

[4.3.5 User Interface – Products 13](#_Toc137856663)

[4.3.5.1.1 Iteration 1 13](#_Toc137856664)

[4.3.5.1.2 Iteration 2 13](#_Toc137856665)

[4.3.6 User Interface – QR code 14](#_Toc137856666)

[4.3.6.1.1 Iteration 1 14](#_Toc137856667)

[4.3.6.1.2 Iteration 2 14](#_Toc137856668)

[4.4 Business Logic Development 15](#_Toc137856669)

[4.4.1 Algorithm 1 - Login 15](#_Toc137856670)

[4.4.1.1.1 Iteration 1 15](#_Toc137856671)

[4.4.1.1.2 Iteration 2 15](#_Toc137856672)

[4.4.2 Algorithm 2 - Register 15](#_Toc137856673)

[4.4.2.1.1 Iteration 1 15](#_Toc137856674)

[4.4.2.1.2 Iteration 2 15](#_Toc137856675)

[4.4.3 Algorithm 2 - Discount 15](#_Toc137856676)

[4.4.3.1.1 Iteration 1 15](#_Toc137856677)

[4.4.3.1.2 Iteration 2 15](#_Toc137856678)

[4.5 Data Access Development 16](#_Toc137856679)

[4.5.1 Database Implementation 16](#_Toc137856680)

[4.5.1.1 Database Creation Class – DatabaseHelper.java 16](#_Toc137856681)

[4.5.2 Data Access Adapters 17](#_Toc137856682)

[4.5.2.1 Accessor Methods 17](#_Toc137856683)

[4.5.2.1.1 Iteration 1 17](#_Toc137856684)

[4.5.2.1.2 Iteration 2 17](#_Toc137856685)

[4.6 Conclusion 17](#_Toc137856686)

[List of Figures 18](#_Toc137856687)

[List of Tables 18](#_Toc137856688)

[References 19](#_Toc137856689)

# 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 | Focus on designing and developing the visual aspects of the application. |
| Lea | Backend Developer | Focus on developing the functional aspects of the application |
| Lucinda | Documentation | Compile project documentation |
| Santana | Testing | Conduct unit testing |

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

The testing plan outlines different types of testing to be conducted at each increment of the app development process. Our team has chosen to focus on unit testing, and this will be conducted at various intervals throughout the applications development to ensure thorough evaluation and improvement of the Sustainable Spoonful app. To assist us with this, we will be using Junit to

Unit testing is a critical testing type that focuses on testing individual units or components of the Sustainable Spoonful app to ensure their correctness and functionality. In this subsection, we will discuss the plans for Unit Testing and design a testing report. We will also outline how we plan to respond to the test results as we continue to design and implement the app. (Contributors, 2023)

We chose to focus on unit testing as it saves time and is easier to work with. Unit testing does not require developers to make changes to their environment to run them nor would we need to incorporate external sources by extension making them quicker and more efficient to work with. (Hamilton, 2023)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Plans** | | | | |
| **Increment** | **Test Type** | **Test Date** | **Team Members** | **…** |
| 1 | Unit Testing | 18/06/2024 | Lucinda, Lea | Testing individual units/components of the app in isolation. |

Table 4.2: Test Plan

(Hamilton, 2023)

Test plan:

1. **Test Coverage***:* We will identify the key units or components of the app that need to be tested. This includes functions, classes, and modules.
2. **Test Cases***:* We will create test cases for each unit, covering different scenarios and edge cases. These test cases will include inputs, expected outputs, and any specific conditions or constraints to be considered.
3. **Test Execution***:* The tester will execute unit tests for the units they are responsible for. This will involve running the test cases and verifying the actual outputs against the expected outputs.
4. **Test Reporting***:* We will document the test results, including any failed test cases, errors encountered, and deviations from the expected behaviour. This information will be captured in a testing report.
5. **Test Result Analysis***:* We will analyse the test results to identify any patterns or common issues. This will help us understand the overall quality and stability of the tested units.
6. **Bug fixing and iteration***:* Based on the test results, any identified issues or bugs will be addressed by the respective developers. The necessary fixes and improvements will be made, and the unit tests will be rerun to validate the changes. (Hamilton, 2023)

Testing Report:

1. **Summary**: A brief overview of the unit testing activities conducted, including the number of units tested, the number of test cases executed, and the overall test coverage achieved.
2. **Test Results***:* A detailed breakdown of the test results, indicating the number of passed and failed test cases, along with any errors or issues encountered during testing.
3. **Deviations***:* Any observed deviations from the expected behaviour or discrepancies in the actual outputs comparted to the expected outputs.
4. **Bug Tracking***:* A list of identified bugs or issues, including their descriptions, severity levels, and assigned developers for resolution.
5. **Recommendations***:* Any recommendations or suggestions for improving the unit testing process or enhancing the quality of the tested units. (Anon, 2023)

\*test report

Responding to Test Results:

Based on the test results obtained from Unit Testing, we will respond in the following ways:

1. **Bug fixing***:* Any identified issues or bugs will be addressed promptly. The respective developers will work on fixing the bugs and ensuring the units meet the desired functionality and quality standards.
2. **Code Refactoring***:* If any areas of the codebase are found to be complex, inefficient, or prone to errors during unit testing, we will consider refactoring those sections to improve code quality and maintainability.
3. **Test Coverage Enhancement***:* If certain units have low test coverage or if important functionalities are not adequately tested, we will update our test plan and add more test cases to improve coverage.
4. **Iterative Testing***:* As the app’s design and implementation progress, we will continue to conduct unit testing iteratively to ensure that new features or modifications do not introduce regressions or impact the existing functionality.

By actively responding to the test results obtained from Unit Testing, we can ensure the identification and resolution of issues at an early stage of development. This approach helps maintain the stability and quality of the Sustainable Spoonful app throughout its development lifecycle.

(Nalawade, 2023)

## Layouts Development

### User Interface – Landing Page

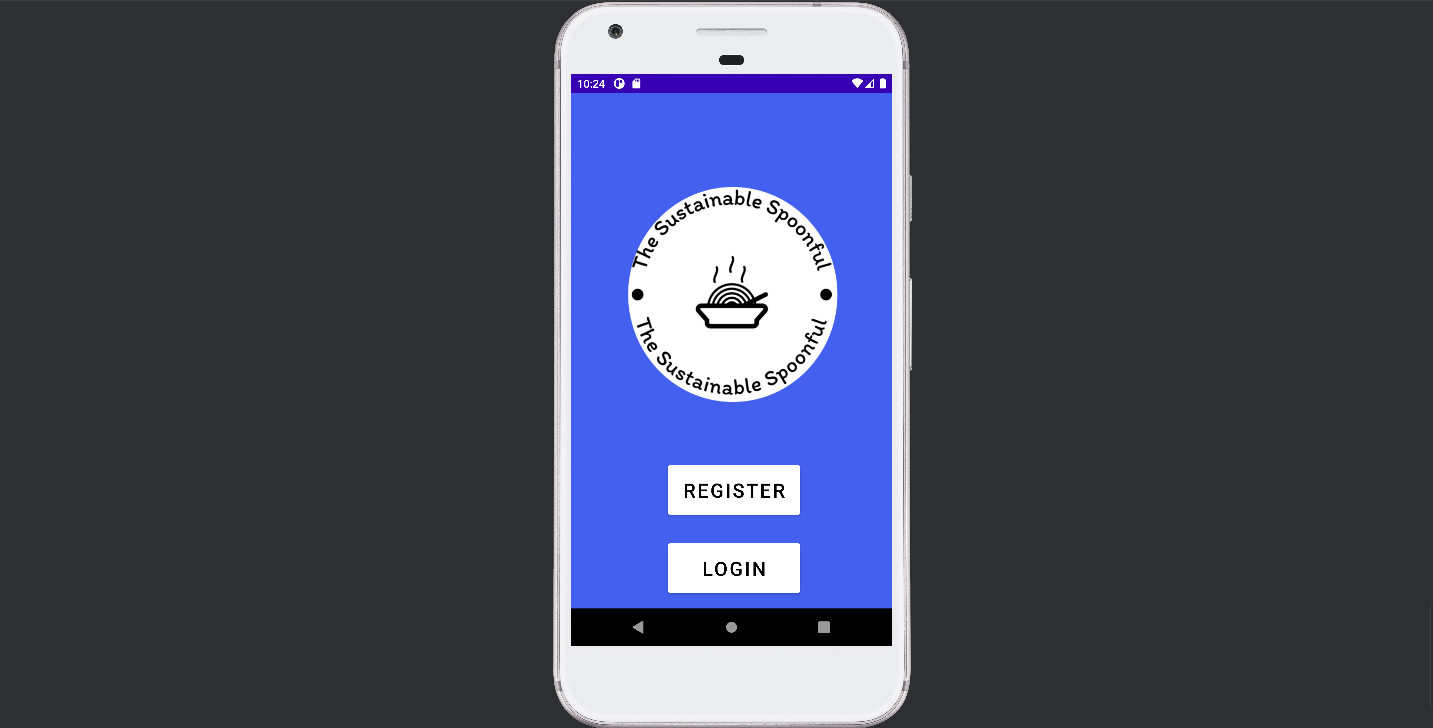
##### Iteration 1

A screenshot of a blue screen

Description automatically generated with low confidence

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

##### Iteration 2



*Figure 2 - Landing Page when the user opens the application for the first time – Emulator PIXEL XL 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.

Core functionality:

public class LandingActivity extends AppCompatActivity {  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_landing*);  
 }  
}

### User Interface - Registration

##### Iteration 1

A screen shot of a phone

Description automatically generated with low confidence

*Figure 3 - Registration Page so that users can create an account.*

##### Iteration 2

Core functionality:

@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_register*);  
  
 //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();  
 }  
 });  
  
}

### User Interface - Login

##### Iteration 1

A screen shot of a phone

Description automatically generated with medium confidence

*Figure 4 - Login Page so that users can login using the account they have just created.*

##### Iteration 2

Core functionality:

private void login(){  
 //Getting the email and password that the customer has entered:  
 String email = emailEditText.getText().toString();  
 String password = passwordEditText.getText().toString();  
  
 //Check if any of the input fields are empty before logging in the customer:  
 if(email.isEmpty()||password.isEmpty()){  
 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){  
 //Displaying a success message:  
 Toast.*makeText*(LoginActivity.this, "Logged in successfully!", Toast.*LENGTH\_SHORT*).show();  
 //Redirecting to the Landing 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:  
 Toast.*makeText*(LoginActivity.this, "Invalid email or password! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }  
  
 //Closing the database:  
 db.close();  
}

### User Interface - Stores

##### Iteration 1

A screen shot of a phone

Description automatically generated with medium confidence

*Figure 5 - Home Page for the Sustainable Spoonful Mobile Application.*

##### Iteration 2

Core functionality:

### User Interface – Products

##### Iteration 1

A screenshot of a cell phone

Description automatically generated with medium confidence

*Figure 6 - Discounts listed for a specific store.*

##### Iteration 2

Core functionality:

### User Interface – QR code

##### Iteration 1

A screen shot of a cell phone

Description automatically generated with medium confidence

*Figure 7 - QR code for the selected discounted product that users can scan at the store to collect their item.*

##### Iteration 2

Core functionality:

## Business Logic Development

### Algorithm 1 - Login

##### Iteration 1

##### Iteration 2

### Algorithm 2 - Registeration

Code Snippet: Inserting new customers details into the database

private void insertCustomer(){  
 //Getting all of the data that was input in the form:  
 String name = nameEditText.getText().toString();  
 String surname = surnameEditText.getText().toString();  
 String email = emailEditText.getText().toString();  
 String password = passwordEditText.getText().toString();  
 String confirmPassword = confirmPasswordEditText.getText().toString();  
  
 //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()){  
 Toast.*makeText*(RegisterActivity.this, "Please fill out all fields in this form.", Toast.*LENGTH\_SHORT*).show();  
 return; //Exit 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){  
 Toast.*makeText*(RegisterActivity.this, "Registration was successful!", Toast.*LENGTH\_SHORT*).show();  
 //Redirect to the login page:  
 startActivity(new Intent(RegisterActivity.this, LoginActivity.class));  
 finish(); //Finishing the current activity so that customers' cannot go back to it when pressing the back button  
  
 }else{  
 Toast.*makeText*(RegisterActivity.this, "Registration failed! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 }else{ //Passwords do not match, display an error message:  
 Toast.*makeText*(RegisterActivity.this, "The passwords entered do not match! Please try again.", Toast.*LENGTH\_SHORT*).show();  
 }  
}

### Algorithm 2 - Discount

##### Iteration 1

##### Iteration 2

## 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:  
 private static final String *DATABASE\_NAME* = "sustainable\_spoonful.db";  
 private static final int *DATABASE\_VERSION* = 1;  
  
 //Providing the table name and column names:  
 public static final String *TABLE\_NAME* = "customer";  
 private static final String *COLUMN\_ID* = "customer\_id";  
 public static final String *COLUMN\_NAME* = "customer\_name";  
 public static final String *COLUMN\_SURNAME* = "customer\_surname";  
 public static final String *COLUMN\_EMAIL* = "customer\_email";  
 public static final String *COLUMN\_PASSWORD* = "customer\_password";  
  
 //Query to create the table:  
 private static final String *CREATE\_TABLE* =  
 "CREATE TABLE " + *TABLE\_NAME* + " (" +  
 *COLUMN\_ID* + " INTEGER PRIMARY KEY," +  
 *COLUMN\_NAME* + " TEXT," +  
 *COLUMN\_SURNAME* + " TEXT," +  
 *COLUMN\_EMAIL* + " TEXT," +  
 *COLUMN\_PASSWORD* + " TEXT)";  
  
 public DatabaseHelper(Context context){  
 super(context, *DATABASE\_NAME*, null, *DATABASE\_VERSION*);  
 }  
  
 @Override  
 public void onCreate(SQLiteDatabase db){  
 //Create the database table:  
 db.execSQL(*CREATE\_TABLE*);  
 }  
  
 @Override  
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){  
 //Upgrade the database if necessary:  
 db.execSQL("DROP TABLE IF EXISTS " + *TABLE\_NAME*);  
 onCreate(db);  
 }  
  
}

### Data Access Adapters

In this subsection, discuss and illustrate the class used to access the data from the database. Snippets of code can be shown together with screenshots of layouts showing the extracted data.

#### Accessor Methods

In this subsection, illustrate the algorithms used to retrieve data from the database. You can show the incremental developed of these algorithms and methods.

##### Iteration 1

##### Iteration 2

## 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 Page when the user opens the application for the first time.* 8](#_Toc137849178)

[*Figure 2 - Landing Page when the user opens the application for the first time – Emulator PIXEL XL API 30* 9](#_Toc137849179)

[*Figure 3 - Registration Page so that users can create an account.* 10](#_Toc137849180)

[*Figure 4 - Login Page so that users can login using the account they have just created.* 11](#_Toc137849181)

[*Figure 5 - Home Page for the Sustainable Spoonful Mobile Application.* 12](#_Toc137849182)

[*Figure 6 - Discounts listed for a specific store.* 13](#_Toc137849183)

[*Figure 7 - QR code for the selected discounted product that users can scan at the store to collect their item.* 14](#_Toc137849184)

# List of Tables

[Table 4.1: Test Plan 3](#_Toc106011508)

# 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.

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/

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].