National College of Ireland

BSHC
Software Development

2022/2023

Shane Mulrooney
x19454016
x19454016@ncirl.ie

MeCeipt
Technical Report

Table of Contents

Executive Summary

This documentation provides insight into the development of my final year project titled MeCeipt. MeCeipt is a receipt distribution app for Android devices which allows users to receive digital receipts instead of traditional, paper-based receipts.

The purpose of this documentation is to provide a background on the purpose and scope of the project, along with providing details of the functional and non-functional requirements gathered to begin developing the app.

The technologies involved with the development of the app are also discussed in this documentation.

1. Introduction

1.1. Background

For the past 8 years, I've worked for one of the largest and fastest growing retail companies in the world. Throughout that time, many decisions have been made by the company and many other retail companies to improve their impact on the environment. A lot of these decisions have been centred around the reduction of paper waste and the implementation of digital technology to replace paper files such as payslips, rosters, and reports.

However, one area I've noticed where the reduction of paper waste hasn't been given such consideration to is the distribution of receipts to customers. After some research, I've found that my company serves approximately 1 million customers per day, worldwide which equates to approximately 1 million paper receipts distributed every single day. Each roll of receipt paper can distribute around 200 receipts and it's been estimated that 55,000 receipts is the equivalent of one tree [1]. This could potentially be an indicator that receipt distribution in my company is responsible for consuming approximately 18 trees every single day.

It is my belief that, although everyone has a responsibility in protecting our planet from environmental issues, it is the large multinational companies that can make the biggest differences and that's why I've chosen MeCeipt as my final year project. MeCeipt can help businesses and consumers visualise the environmental benefits of transitioning to a fully digital receipt distribution system.

1.2. Aims

The aim of this project is to inform customers and businesses of the economic and environmental impacts of physical receipt distribution and provide visualisation of such impacts. I want customers and business to feel empowered by the environmentally friendly decisions they make by choosing MeCeipt as their primary method of receipt distribution and management and avoiding what I consider to be an obsolete and costly method in the form of traditional paper receipt distribution.

MeCeipt will allow registered customer users to receive a digital receipt at a store's point-of-sale area instead of a traditional paper receipt. The receipts that customer users accumulate will be viewable and manageable from a screen within the app. Customer users will also be able to view the environmental statistics collected as a result of using MeCeipt.

The app will also allow registered business users to view the environmental statistics they have collected as a result of their customers using MeCeipt. These statistics are in an effort to inform businesses and customers the environmental benefits of replacing paper receipts with digitalised ones.

2. Technology

There have been several technologies used for the implementation of my project up until this point. Below is an overview of the technologies used as of now:

Technology	Function
Android Studio	**Development Environment**
Kotlin	**Programming Language**
XML	**Markup Language for Styling in Android Studio**
Firebase	**User Authentication**
Firestore	**Database**

2.1. Android Studio

Android Studio is my IDE of choice for this project. Google heavily recommends Android Studio for the development of Android apps because of its easy-to-use nature and ideal implementation of various modules. My hope is that, at some point, MeCeipt would be available to over 95% of mobile device users to minimise the environmental impacts of paper receipt distribution, however, given the minimal time available to me to learn new technologies, I have decided, for this project, to develop my app for Android devices. Android Studio allows my app to be compatible with over 98% of Android devices, maximising the Android userbase.

2.2. Kotlin

Android Studio has gradually migrated from Java being the primary programming language to Kotlin. Although I have more experience with Java, it is Google's recommendation to program Android apps with Kotlin as it has become more supported for Android Studio.

2.3. XML

Android Studio uses XML for the styling of activities for Android app development. This intuitive way of styling apps is both easy to learn and offers efficient load times for users.

2.4. Firebase

User Authentication is an imperative functionality for my project. After some research, I discovered that Google's Firebase offers an easy-to-implement, user-friendly User Authentication system. Using Firebase's UI, I can easily manage users registered to my application.

2.5. Firestore

Google's Firebase offers various databases for storing data. The one I have chosen for this project is Firestore. Firestore provides a many-to-many infrastructure and is quite scalable, making for an ideal database for my project.

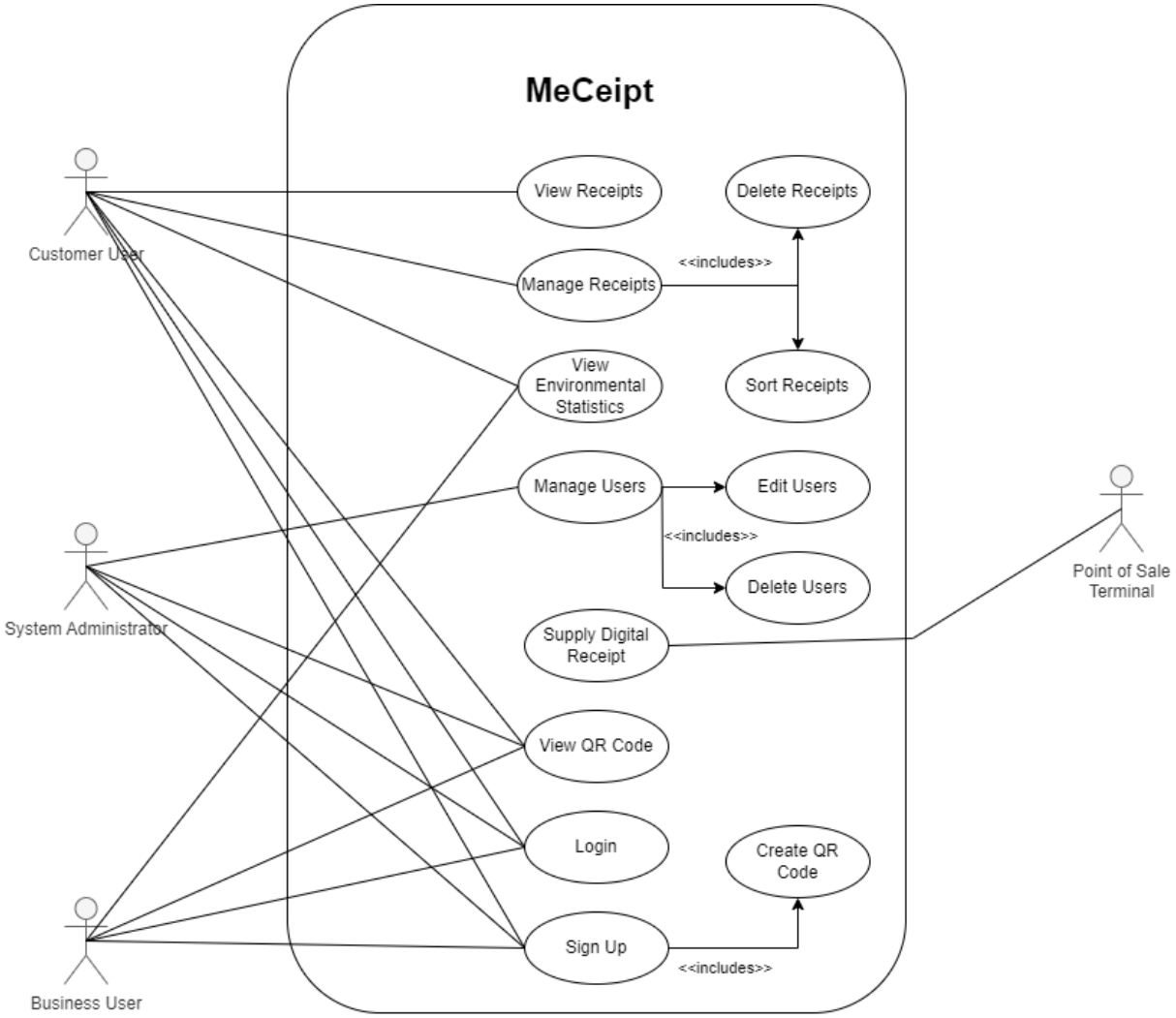
3. Structure

The structure of this document is as follows:

- **Section 4 –** This section provides the Software Requirement Specification for my application. In this section you can find the Use Case Diagram and all of the functional and non-functional requirements along with their corresponding use cases for my application.
- **Section 5 –** This is the Design & Architecture section. Here you will find high-level diagrams of the architecture of the technologies used to develop my application along with a high-level structure of the database design.
- **Section 6 –** This is the Implementation section. Here you will find explanations and code snippets for some of the most interesting and unique pieces of implementation in my application.
- **Section 7 –** In this section you will find all of the screenshots of the Graphical User Interface for my application.
- **Section 8 –** This is the Testing & Evaluation section. Here you will find evidence of testing within my application and a table containing acceptance tests based on the requirements specified in the Software Requirement Specification in Section 4.

4. Requirement Specification

4.1. Use Case Diagram



4.2. Functional Requirements

4.2.1. User Registration

- **Description & Priority**

This requirement involves the registration of a user to the system. This requirement is of crucial importance as, in order for users to gain access to the application's main functionalities, they must create an account by registering.

- **Use Case**

Description/Scope	Allows a customer user or business user to register with the application and create an account to access application functionality.

Pre-Condition	User is on 'Welcome Screen' and has not yet created an account.
Activation	User selects 'Sign Up' button from 'Welcome Screen'.
Main Flow	1. User visits 'Sign Up Screen' 2. User enters information in the required fields 3. Application informs user of successful registration
Alternate Flow	Required fields not filled: 1. User has not filled out the required fields 2. Application informs user to fill out required fields User already registered: 1. User registers with an email already registered 2. Application informs user that the user has already registered with that email address
Termination	• Main Flow: User registers successfully. • Alternate Flow: User fails to register.
Post Condition	• Main Flow: User transferred to 'Main Screen'. • Main Flow: User must attempt to register again.

4.2.2. User Login

- **Description & Priority**

This requirement involves a registered user logging into the system to access the functionalities of the application. This requirement is of crucial priority as only a logged-in user should gain access to the applications' functionalities.

- **Use Case**

Description/Scope	Allows a registered user to log in to the application to access its functionalities.
Pre-Condition	User is registered on the system and is on 'Welcome Screen'.
Activation	Registered user selects 'Sign In' button from 'Welcome Screen'.
Main Flow	1. Registered user visits 'Sign In Screen' 2. User enters login credentials i.e., email address and password 3. Application informs user of successful login
Alternate Flow	Incorrect email address and/or password: 1. User enters incorrect email address or password in respective fields 2. Application informs user of incorrect email and/or password User doesn't exist: 1. Unregistered user attempts to login using unregistered credentials 2. Application informs user to sign up
Termination	• Main Flow: User logs in successfully. • Alternate Flow: User fails to login.
Post Condition	• Main Flow: User transferred to 'Main Screen'. • Main Flow: User must attempt to login again.

4.2.3. Personalised Unique QR Code

- **Description & Priority**

This requirement involves the automatic generation of a personalised QR code when a user has created their account. This requirement is of crucial priority as a QR code must be used to gather receipts from a point-of-sale terminal.

➢ **Use Case**

Description/Scope	Generates a personalised QR code for registered users.
Pre-Condition	User has entered valid credentials on 'Sign Up Screen'.
Activation	User selects 'Sign Up' button on 'Sign Up Screen'.
Main Flow	1. QR code generated upon user registering. 2. QR code visible from 'Main Screen'.
Alternate Flow	QR code fails to generate: 1. User registers with the system. 2. Error occurs in generation of QR code. 3. User requested to return at a different time. QR code fails to display: 1. Registered user navigates to 'Main Screen'. 2. Error occurs in displaying QR code. 3. User requested to refresh the app.
Termination	• Main Flow: QR code successfully generates and displays. • Alternate Flow: QR code fails to generate and/or display.
Post Condition	• Main Flow: QR code visible from 'Main Screen'. • Alternate Flow: User must attempt to sign up again or refresh application.

4.2.4. View Receipts

➢ **Description & Priority**

This requirement involves a logged-in user viewing the receipts they have accumulated. This requirement is of crucial priority as user's must be able to see their receipts to use them.

➢ **Use Case**

Description/Scope	Allows user to view the receipts they have collected.
Pre-Condition	User is logged in and has accumulated at least 1 receipt.
Activation	User navigates to 'Receipt Screen' from 'Main Screen'.
Main Flow	1. Logged-in user navigates to 'Receipt Screen'.
Alternate Flow	No receipts accumulated: 1. User navigates to 'Receipt Screen' despite having accumulated zero receipts.
Termination	• Main Flow: Accumulated receipts are displayed. • Alternate Flow: Application informs user that they have accumulated 0 receipts.
Post Condition	• Main Flow: Receipts visible from 'Receipt Screen' • Alternate Flow: No receipts visible from 'Receipt Screen'

4.2.5. Accumulate Receipt

➢ **Description & Priority**

This requirement involves a user scanning their personalised QR code at a point-of-sale terminal to accumulate a receipt. This requirement is of crucial priority as it's one of the application's main functionalities.

- **Use Case**

Description/Scope	Allows user to accumulate receipt at a point-of-sale terminal using QR code.
Pre-Condition	User has made a purchase at a store and opted for digital receipt.
Activation	User scans QR code at point-of-sale terminal.
Main Flow	1. User navigates to 'Home Screen' 2. User scans QR code at terminal 3. Receipt appears on 'Receipt Screen'
Alternate Flow	N/A
Termination	Receipt has appeared on 'Receipt Screen'
Post Condition	New receipt is visible to user from 'Receipt Screen'

4.2.6 Manage Receipts

- **Description & Priority**

  This requirement involves a logged-in user deleting the receipts they have accumulated. This requirement is of medium priority as, although it is not an imperative part of the overall functionality, it offers users a user-friendly user-interface.

- **Use Case**

Description/Scope	Allows user to delete or sort receipts accumulated from the 'Receipt Screen'.
Pre-Condition	User is logged in and has accumulated at least 1 receipt.
Activation	User selects 'Manage Receipts' button from 'Receipt Screen'
Main Flow	1. Logged-in user navigates to 'Receipt Screen' 2. User selects 'Manage Receipts' button
Alternate Flow	<u>Delete Receipts</u> 1. Logged-in user navigates to 'Receipt Screen' 2. User selects receipt(s) to delete 3. Application requests confirmation from user to delete selected receipts
Termination	• Main flow: User on 'Manage Receipts' screen • Alternate Flow: User has deleted receipts
Post Condition	• Main Flow: User can select how they wish to manage receipts • Alternate Flow: Deleted receipts no longer visible

4.2.7. View Statistics

- **Description & Priority**

  This requirement involves a business user or a customer user viewing the environmental statistics that have been collected by using MeCeipt. This requirement is of high priority as one of the main purposes of the application is to inform users of the environmental benefits digital receipt distribution and accumulation has over traditional paper receipts.

➢ **Use Case**

Description/Scope	Allows business user or customer user to view environmental statistics.
Pre-Condition	• Customer user has accumulated at least 1 receipt • Business user has signed up to the system
Activation	User navigates to 'Statistics Screen'
Main Flow	Customer user navigates to 'Statistics Screen'
Alternate Flow	Business User: Business user navigates to 'Statistics Screen'
Termination	• Main Flow: Customer user is on 'Statistics Screen' • Alternate Flow: Business user is on 'Statistics Screen'
Post Condition	• Main Flow: Customer user can view personalised statistics • Alternate Flow: Business user can view business statistics

4.2.8. Manage Users

➢ **Description & Priority**

This requirement involves an admin user removing users from the system. This requirement is of medium priority.

➢ **Use Case**

Description/Scope	Allows admin user to add or remove users from system.
Pre-Condition	Admin user is logged-in
Activation	Admin user navigates to 'Manage Users Screen'
Main Flow	1. Admin user is logged in 2. User navigates to 'Manage Users Screen'
Alternate Flow	N/A
Termination	Admin user is on 'Manage Users Screen'
Post Condition	Admin user can delete users to and from system.

4.3. Non-Functional Requirements

4.3.1. Accessibility Requirements

There are a number of requirements to be considered when developing this application with accessibility in mind. MeCeipt has been designed so that visually impaired or colour-blind users can as easily navigate the app as users without such an impairment. MeCeipt's black and white User Interface and larger buttons and UI elements helps to accomplish this requirement.

4.3.2. Data Requirements

The database of choice for this project is Firebase's Firestore. This database was chosen due to Google's recommendation for data storage with Android app development and integration with Android Studio.

4.3.3. Security Requirements

It is imperative that user data is stored in a safe and secure manner, therefore it is required that only essential data is stored and sensitive data such as passwords are encrypted. It is also important that proper authentication and authorization is implemented so that users can only access their own data.

5. Design & Architecture

5.1. Technology Architectural Diagram

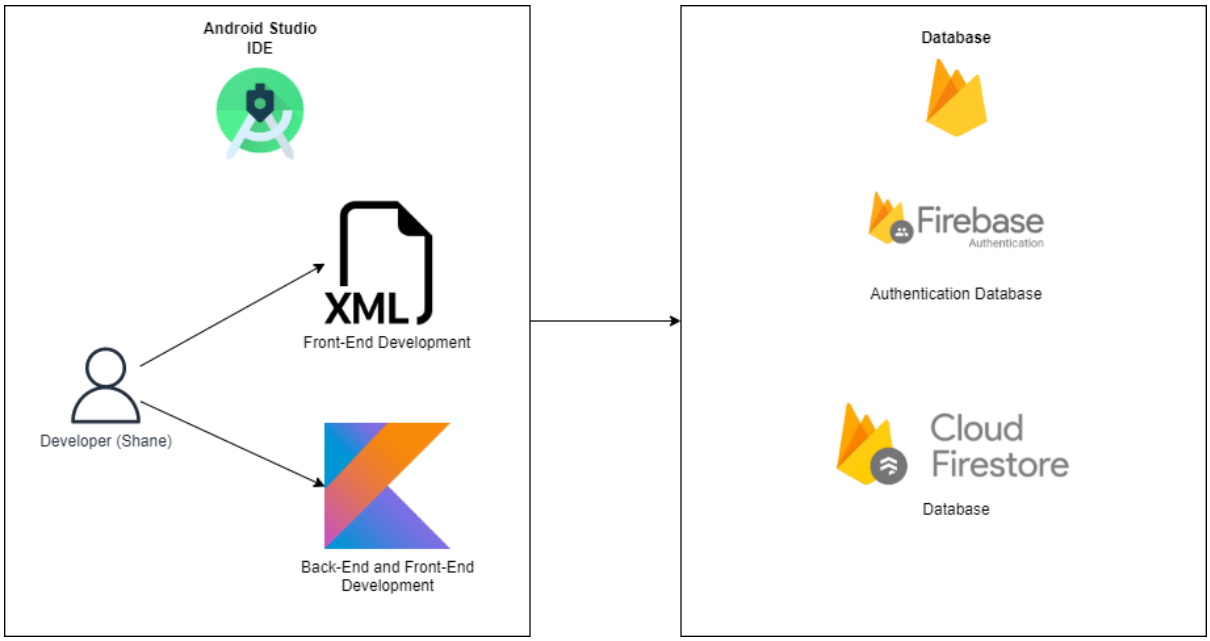


Figure 1

Figure 1 shows the architectural diagram for the technologies I used to develop this project.

5.2. Firebase Firestore Database

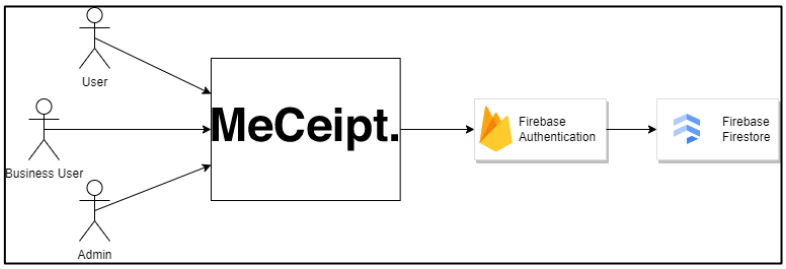


Figure 2

Figure 1 shows the architecture for the Firebase Firestore database used to store all of the data for the project. The data stored includes user data and their corresponding receipt data, business user data and admin user data.

Identifier	Providers	Created ↓	Signed in	User UID
test@meceipt.com	✉	26 Apr 2023	26 Apr 2023	hPp6QGX39QWPLtjzJ9YWRWu1V...
test1@email.com	✉	20 Apr 2023	20 Apr 2023	hKQAJ7pG1fhsv6ApiLjgipuuhHl2
admin@email.com	✉	18 Apr 2023	20 Apr 2023	C0QgNNf96ITlOrqV3Mbt6lCcxHz1
test@email.com	✉	4 Apr 2023	3 May 2023	VwExvDKG4lYlr6MUAOJpQVxx51c2
penneys@meceipt.com	✉	14 Mar 2023	2 May 2023	beLCZJBREdVQQF3iDhhm3hErUZ...

Figure 3

Figure 2 shows the Firebase Authentication database where user data is safely and securely stored.

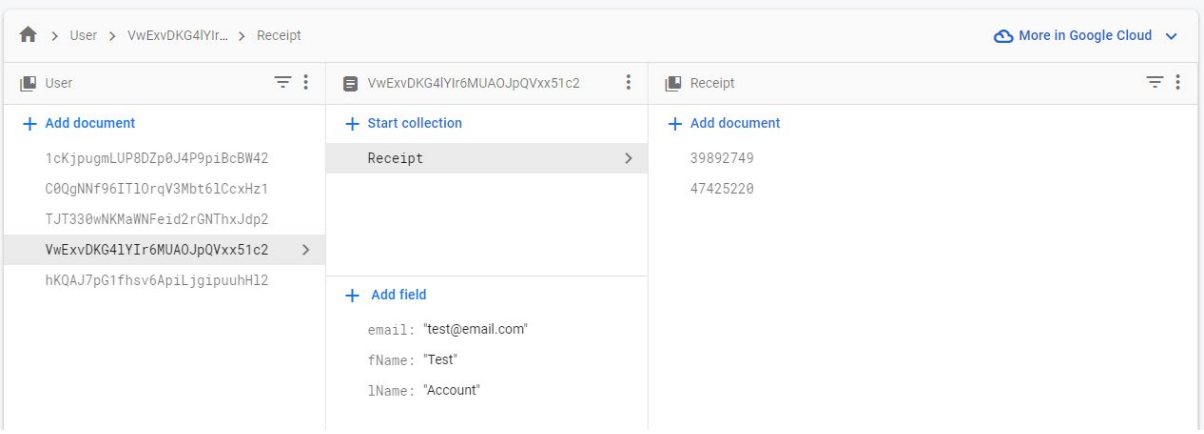


Figure 4

Figure 3 shows the Firebase Firestore collection containing documents for User data and their corresponding Receipt documents.

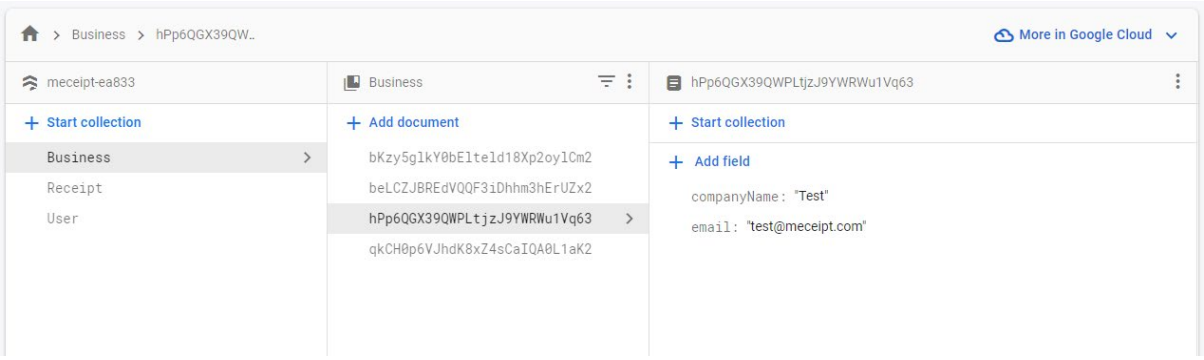


Figure 5

Figure 4 shows the Firebase Firestore collection for containing documents for Business User data.

5.3. Continuous Integration

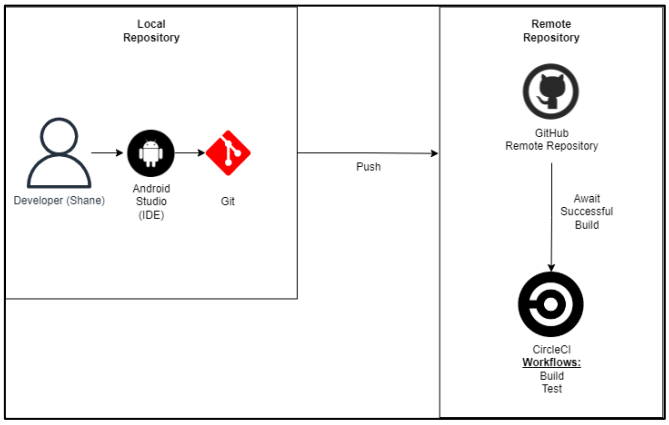


Figure 6

Figure 3 shows the architecture for the Continuous Integration pipeline for my project.

6. Implementation

In this section, I will provide insight into the implementation of the most interesting and unique elements of my application.

6.1. User and Business User Registration

The implementation of User and Business User registration is handled primarily by Firebase Authentication. The implementation of the registration of a Business User differs slightly from a regular User.

```
firebaseAuth = FirebaseAuth.getInstance()

binding.btnSignUpPage.setOnClickListener {  it: View!
    val companyName = binding.tfCompany.text.toString().trim()
    val password = binding.tfPassword.text.toString().trim()
    val confPassword = binding.tfConfPassword.text.toString().trim()

    if (companyName.isNotEmpty() && password.isNotEmpty() && confPassword.isNotEmpty()) {
        if (password != confPassword) {
            Snackbar.make(view,  text: "Passwords do not match", Snackbar.LENGTH_LONG).show()
        } else {
            val emailLower = "$companyName@meceipt.com".lowercase()

            firebaseAuth.createUserWithEmailAndPassword(emailLower, password).addOnCompleteListener{  it: Task<AuthResult!>
                if (it.isSuccessful){
                    val business = hashMapOf(
                        "companyName" to companyName,
                        "email" to emailLower
                    )
                    val currentUser = FirebaseAuth.getInstance().currentUser
                    val userId = currentUser!!.uid
                    val businessDocRef = firestore.collection( collectionPath: "Business").document(userId)
                    businessDocRef.set(business).addOnSuccessListener {  it: Void!
                        Snackbar.make(view,  text: "Business User Created", Snackbar.LENGTH_LONG).show()
                        val businessHome = Intent( packageContext: this, BusinessHomeActivity::class.java)
                        startActivity(businessHome)

                    }

                }
            }

        }
    }
}
```

Figure 7

The code snippet above shown in Figure 4 is for the registration of Business Users. I wanted a Business User to have a unique MeCeipt email address upon registration and in order to do this, I created a variable that interpolated a string from the name of the company the user inputted upon registering and a string “@meceipt.com”. I then created a hash map for the company name and email that can be added to a collection called “User” in a document with the ID of the corresponding Firebase Authentication user.

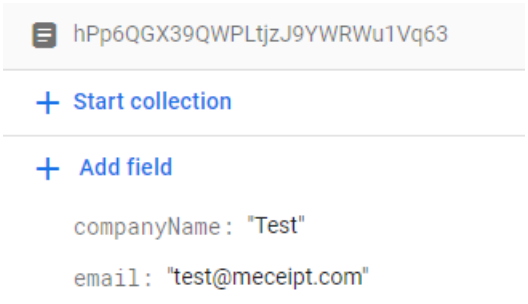


Figure 8

Figure 5 above shows a Firestore document created from a Business User registering with the company name: Test.

```
binding.btnSignUpPage.setOnClickListener {
    val fName = binding.tfFName.text.toString().trim()
    val lName = binding.tfLName.text.toString().trim()
    val email = binding.tfEmail.text.toString().trim()
    val password = binding.tfPassword.text.toString().trim()
    val confPassword = binding.tfConfPassword.text.toString().trim()

    if (fName.isNotEmpty() && lName.isNotEmpty() && email.isNotEmpty() && password.isNotEmpty() && confPassword.isNotEmpty()){
        if (password != confPassword) {
            Snackbar.make(view, text: "Passwords do not match", Snackbar.LENGTH_LONG).show()

        } else if (email.contains( other: "@meceipt.com", ignoreCase = true)) {
            Snackbar.make(view, text: "Email address cannot be an MeCeipt email", Snackbar.LENGTH_LONG).show()

        } else {
            firebaseAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener{
                if (it.isSuccessful){
```

Figure 9

Figure 6 above is a code snippet of the User registration function. Firstly, I implemented some validations to ensure the fields are not empty and that password and confirm password fields match. I also check to make sure the email address of the registering User does not contain the string "@meceipt.com".

```
if (it.isSuccessful){

    val currentUser = FirebaseAuth.getInstance().currentUser
    val userId = currentUser!!.uid
    val userDocRef = firestore.collection( collectionPath: "User").document(userId)

    val user = hashMapOf(
        "fName" to fName,
        "lName" to lName,
        "email" to email
    )

    userDocRef.set(user).addOnSuccessListener {

        val home = Intent( packageContext: this, HomeActivity::class.java)
        startActivity(home)
    }
```

Figure 10

Upon successful registration, a hash map of the User data is created and set to a document with the ID of the User's Firebase Authentication ID within the "User" Firestore collection as shown in the code snippet in Figure 7 above.

6.2. Personalised Unique QR Code

The implementation of this functional requirement involved the use of some external libraries available in Android Studio. After some research. I discovered there are a number of QR code generator libraries available, however, I decided to choose ZXing as it suited my need to pass a variable into the bit matrix.

```
import com.google.zxing.BarcodeFormat
import com.google.zxing.qrcode.QRCodeWriter
```

Figure 11

To generate the QR code, I initialised some variables for the user ID of the current logged-in user, the height, and the width of the QR code.

```
val qrText = userId.toString()
val qrCodeWidth = 812
val qrCodeHeight = 812
```

Figure 12

Then I initialised a QR code writer, a bit matrix and a bitmap for the QR code to generate from.

```
val qrCodeWriter = QRCodeWriter()
val bitMatrix = qrCodeWriter.encode(qrText, BarcodeFormat.QR_CODE, qrCodeWidth, qrCodeHeight)
val qrCodeBitmap = Bitmap.createBitmap(qrCodeWidth, qrCodeHeight, Bitmap.Config.RGB_565)
```

Figure 13

To draw the generated QR code on the layout fragment, I implemented the following for loop and set the resulting bitmap to a layout ImageView element named qrCode as shown below.

```
for (x in 0 ≤ until < qrCodeWidth) {
    for (y in 0 ≤ until < qrCodeHeight) {
        qrCodeBitmap.setPixel(x, y, if (bitMatrix[x, y]) Color.BLACK else Color.WHITE)
    }
}

qrCode.setImageBitmap(qrCodeBitmap)
```

Figure 14

6.3. Accumulate Receipt

For the purpose of demonstration and due to the fact that I do not have access to a point-of-sale system and QR code scanner, I have slightly altered the scope of this requirement. This use case requires both a Business User and a regular User to be logged into the system. To accumulate a receipt, a User must scan their QR code at a point-of-sale system but for the purpose of demonstration, the logged in Business User will act as the QR code scanner. Upon logging in, the Business User can tap a button that will open a barcode scanner as shown in the code snippet in Figure 12.

```
val scanBtn = binding.btnScan
scanBtn.setOnClickListener {  it: View!
    val integrator = IntentIntegrator( activity: this)
    integrator.initiateScan()
}
```

Figure 15

The barcode scanner will check if the contents of the QR code matches the document ID of the corresponding User in the Firestore “User” collection.

```
val result = IntentIntegrator.parseActivityResult(requestCode, resultCode, data)
if (result != null) {
    if (result.contents == null) {

    } else {
        val scannedResult = result.contents
        val userCollectionRef = FirebaseFirestore.getInstance().collection( collectionPath: "User")
        val userDocumentRef = userCollectionRef.document(scannedResult)

        userDocumentRef.get().addOnSuccessListener { documentSnapshot ->
            if (documentSnapshot.exists()) {
```

Figure 16

Upon a successful check, a reference to the “Receipt” subcollection is made from the “User” collection with the matching User ID. A new receipt data object is created with some sample data for demonstration purposes.

```
docRef.get().addOnSuccessListener { businessDocumentSnapshot ->
    val userReceiptColRef = FirebaseFirestore.getInstance().collection( collectionPath: "User").document(scannedResult).collection( collectionPath: "Receipt")
    val name = businessDocumentSnapshot.getString( field: "companyName").toString()

    val transaction = randomNumber()

    val currentDate = LocalDate.now()
    val format = DateTimeFormatter.ofPattern( pattern: "dd-MM-yyyy")
    val date = currentDate.format(format)
    val totalCost = randomDouble()

    val  newReceipt = hashMapOf(
        "companyName" to name,
        "address" to null,
        "transactionNumber" to transaction,
        "date" to date,
        "totalCost" to totalCost
    )
```

Figure 17

This new receipt object is then added to the “Receipt” subcollection with an ID of the transaction number as shown in Figure 15:

```
val documentId = transaction.toString()

userReceiptColRef.document(documentId).set(newReceipt).addOnSuccessListener {  it: Void!
    val receipt = Intent( packageContext: this, BusinessHomeActivity::class.java)
    startActivity(receipt)
    Toast.makeText( context: this,  text: "Receipt Added!", Toast.LENGTH_SHORT)
        .show()
    receiptDocRef.document(documentId).set(basicReceipt)
}
```

Figure 18

6.4. View Receipts

This functionality is one of the most crucial and time-consuming implementations of the project. It involves a User viewing the receipts they have accumulated after scanning their QR code. To begin, I

had to create a RecyclerView layout element on the Fragment I want the receipts to be displayed on and a separate layout file containing the views for the individual receipt items I want to display within the RecyclerView.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/userReceiptRecyclerView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Figure 19

```
<TextView
    android:id="@+id/tvCompanyName"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.007"
    android:textColor="@color/black"
    android:textStyle="bold"/>

<ImageButton
    android:id="@+id/btnDelete"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="16dp"
    android:background="@drawable/ic_baseline_delete_24"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/tvDate"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tvCompanyName"
    app:layout_constraintVertical_bias="0.0"
    android:textColor="@color/black"
    android:textStyle="italic"/>
```

Figure 20

In order to use a RecyclerView, I had to create an Adapter and View Holder class. The View Holder is an inner class in the Adapter class which holds references to the views within the individual receipt

item layout file. Within the View Holder class I have a method that binds the data from the User Receipt model to these views.

```
inner class UserReceiptViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    private val binding = UserReceiptItemBinding.bind(itemView)
    private val companyNameTv = binding.tvCompanyName
    private val dateTv = binding.tvDate
    private val deleteBtn = binding.btnDelete


    fun bind(userReceipt: UserReceipt) {
        companyNameTv.text = userReceipt.companyName
        dateTv.text = userReceipt.date.toString()
```

Figure 21

The Adapter class connects the data source to the RecyclerView and is done so by the code provided in Figure 22. Further explanation of this code is provided in my presentation video.

```
class UserReceiptAdapter(private val receiptList: List<UserReceipt>) : RecyclerView.Adapter<UserReceiptAdapter.UserReceiptViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UserReceiptViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.user_receipt_item, parent, attachToRoot: false)
        return UserReceiptViewHolder(view)
    }

    override fun onBindViewHolder(holder: UserReceiptViewHolder, position: Int) {
        holder.bind(receiptList[position])
    }

    override fun getItemCount(): Int {
        return receiptList.size
    }
```

Figure 22

Within the bind function is a an OnClickListener method that executes code when the Company Name view is tapped by the user. The executed code creates a reference to a User Receipt document in a Receipt subcollection of the User collection in Firestore. This reference selects the receipt based on the transaction number from the selected receipt. Upon the success of retrieving the correct receipt document, an alert dialog box is displayed which acts as a modal displaying the expanded receipt data that is not visible from the 'View Receipt Screen'. The data that is shown is an interpolated string containing the data from the retrieved receipt from the database. The code is shown below in Figure 23.

```
val transactionNumber = userReceipt.transactionNumber.toString()

companyNameTv.setOnClickListener {  it: View!
    val user = FirebaseAuth.getInstance().currentUser?.uid.toString()
    val documentRef = FirebaseFirestore.getInstance().collection( collectionPath: "User").document(user).collection( collectionPath: "Receipt").document(transactionNumber)

    documentRef.get().addOnSuccessListener { documentSnapshot ->
        val name = documentSnapshot.getString( field: "companyName")
        val date = documentSnapshot.getString( field: "date")
        val totalCost = documentSnapshot.getDouble( field: "totalCost").toString()
        val transNumber = documentSnapshot.get("transactionNumber")

        val receiptDialogBuilder = AlertDialog.Builder(itemView.context)
        receiptDialogBuilder.setTitle("Receipt from $name on $date")
        receiptDialogBuilder.setMessage("Store: $name\n" +
                "Date: $date\n" +
                "Transaction No.: $transNumber\n" +
                "Total Cost: €$totalCost")
        receiptDialogBuilder.setPositiveButton( text: "OK") { dialog, _ ->
            dialog.dismiss()

        }
        val receiptDialog = receiptDialogBuilder.create()
        receiptDialog.setOnShowListener {  it: DialogInterface!
            receiptDialog.getButton(AlertDialog.BUTTON_POSITIVE)?.setTextColor(ContextCompat.getColor(itemView.context, R.color.black))
        }
        receiptDialog.show()


    }
```

Figure 23

7. Graphical User Interface

This section will outline the front-end of the application. Each screen of the application will be demonstrated, and the functionalities will be highlighted. Screenshots and wireframes will be provided for visualisation purposes.

7.1. Welcome Screen

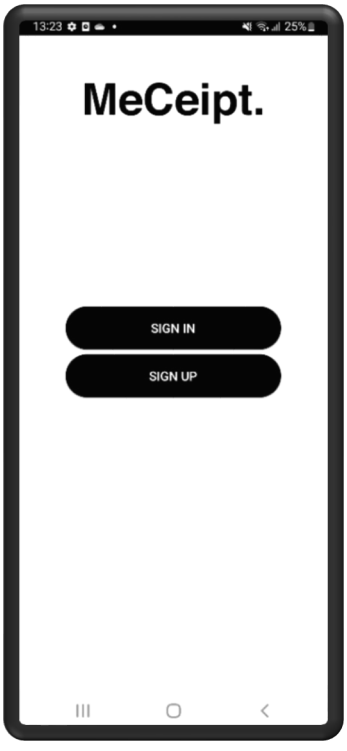


Figure 24

This is the ‘Welcome Screen’ of the application. This is the screen that displays when a user runs the app. This screen has multiple UI elements including 2 interactive buttons which allows navigation to the ‘Sign Up Screen’ and ‘Sign In Screen’.

7.2. Sign In Screen

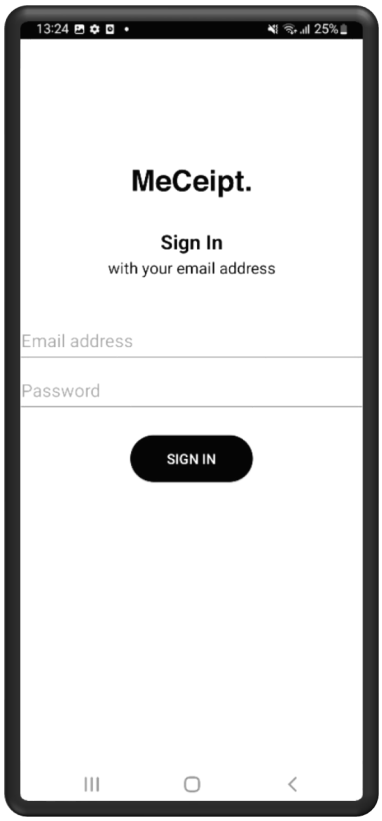


Figure 25

This is the ‘Sign In Screen’. This screen allows registered users to sign into the system by inputting the credentials they chose during registration into the fields provided.

7.3. Sign Up Screen

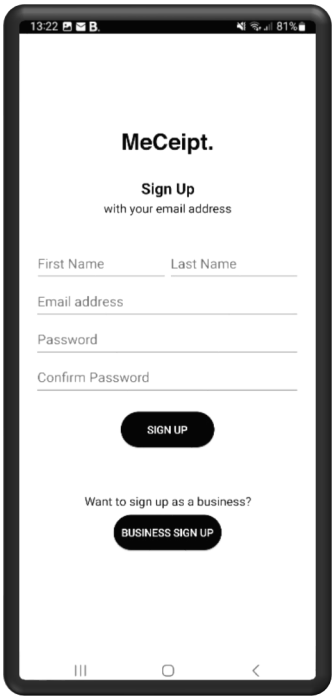


Figure 26

This is the 'Sign Up Screen'. This screen is where users will navigate to in order to register to the system. A user must fill in all the fields provided and tap the 'Sign Up' button to register. Upon registration, the user will be automatically navigated to the 'Home Screen'. This screen also provides a navigation button to a business registration screen via the 'Business Sign Up' button.

7.4. Business Sign Up Screen

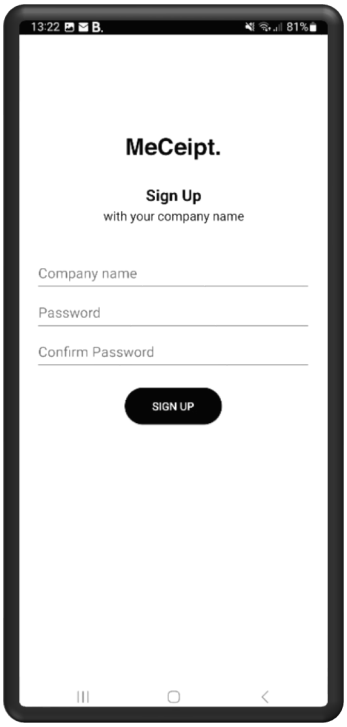


Figure 27

This is the ‘Business Sign Up Screen’. This screen allows Business Users to sign up using their company name a password. The Business User must fill all fields and select the ‘Sign Up Button’ to register. Successful registration will transfer the Business User to the ‘Business Home Screen’.

7.5. Home Screen

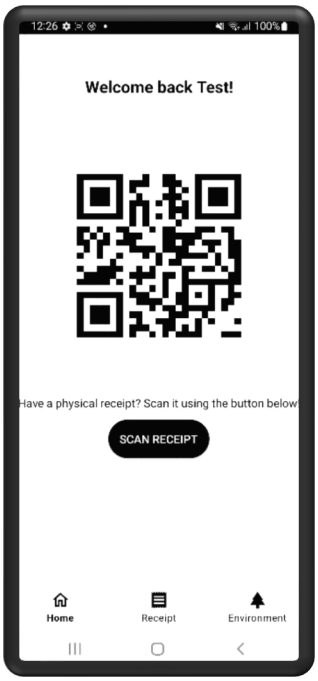


Figure 28

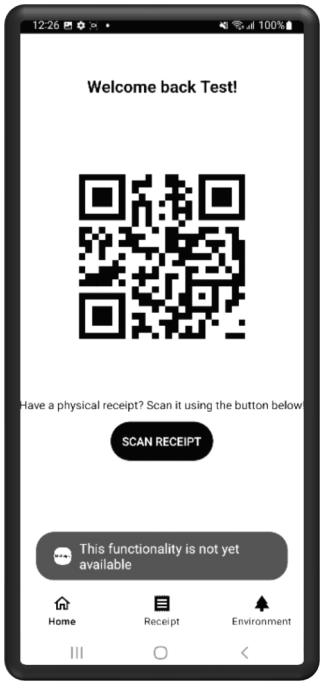


Figure 21

This is the ‘Home Screen’. This where the User can navigate the functionality of the app using the navigation menu at the bottom. This screen shows the User’s personalised QR code which is used to retrieve receipts. This screen also has a button which is used to scan physical paper receipts, however this has not been implemented yet and may be considered for a future iterations as mentioned in the Further Development & Research section.

7.6. Receipt Screen

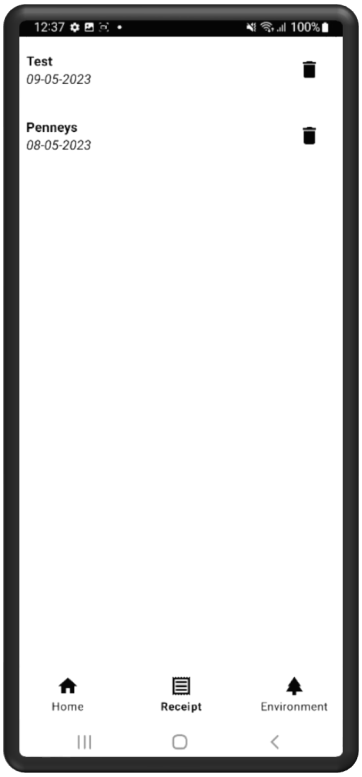


Figure 22

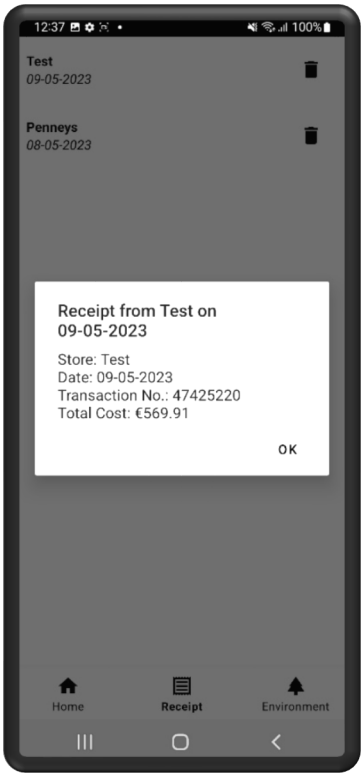


Figure 23

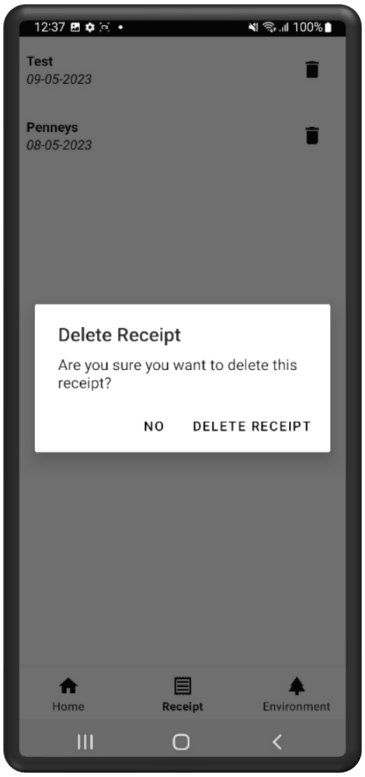


Figure 24

This is the 'Receipt Screen'. This is where the User can view and manage the receipts the have accumulated. The receipts are displayed as a list and each one can be tapped to open an expanded version of the receipt which displays more data. Each receipt also has a delete button which when pressed, opens a confirmation dialog box asking the User if they wish to proceed with the deletion.

7.7. Statistics Screen

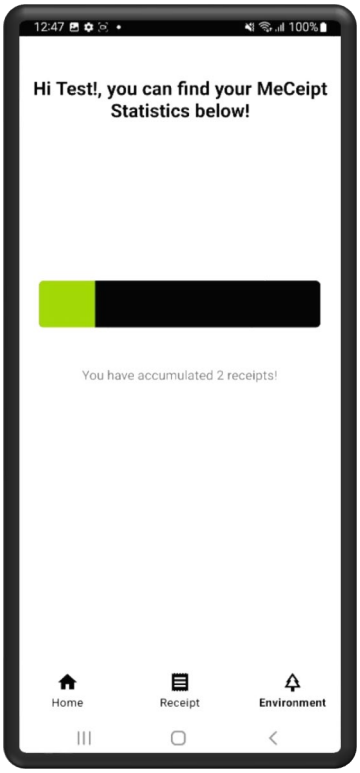


Figure 25

This is the 'Statistics Screen'. This screen displays a progress bar illustrating the number of receipts the User as accumulated. Further implantation might include a potential rewards system which is discussed in the Further Development & Research section.

7.8. Admin Screen and Users Screen

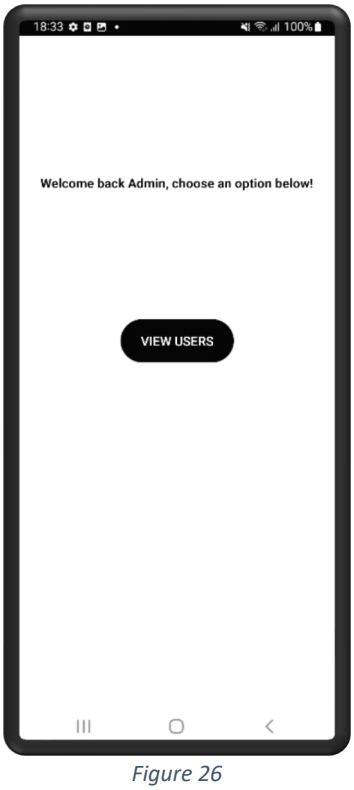


Figure 26

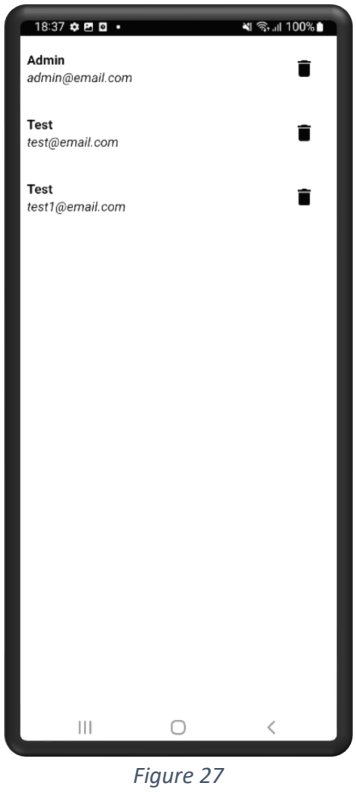


Figure 27

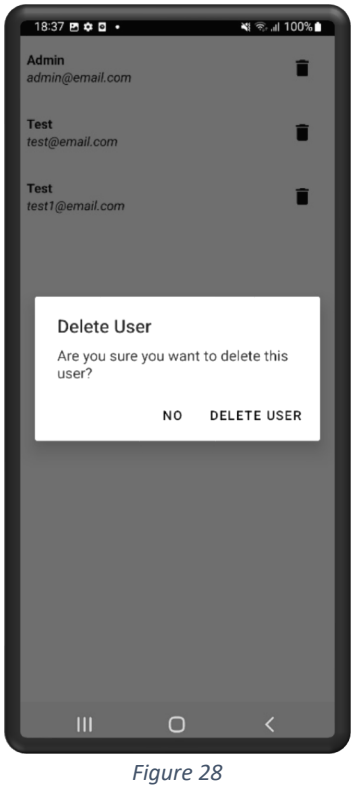


Figure 28

This is the 'Admin Screen' and 'Users Screen'. From this screen, an authorised and authenticated administrative user can access administrative privileges. At this stage in development, admin users can view and delete registered users by accessing the 'Users Screen' via the 'View Users Button' on the 'Admin Screen'.

8. Testing & Evaluation

The testing of my application consisted of instrumented tests in Android Studio which are a type of integration test. These instrumented tests are conducted using various Android Studio and Java libraries. The library I used to conduct my instrumented tests is JUnit. JUnit is a Java library for conducting tests for Java applications that follows the principles of unit testing and provides utilities that support the writing and running of tests for applications in, in my case, Kotlin and Android Studio.

Instrumented testing in Android Studio consists of running the tests in a real-life scenario on either an emulator running a specific version of Android and hardware or on an actual device running a specific version of Android. Due to hardware limitations on my PC, it was not optimal for me to test on an emulator and instead, I used the Wireless and USB Debugging method on my physical Android device running Android 12.

Below I will provide an example of an instrumented test I have implemented and explain the code.

```
@RunWith(AndroidJUnit4::class)
class SignUpSignInTest {
    @get:Rule
    var welcomeActivityRule = ActivityScenarioRule(WelcomeActivity::class.java)

    @Test
    fun signInTest() {
        onView(withId(R.id.btnSignIn)).perform(click())
        onView(withId(R.id.tfEmailSignIn)).check(matches(isDisplayed()))

        val email = "test@email.com"
        val password = "123456"
        val confPassword = "123456"

        FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password).addOnCompleteListener() { task ->
            assert(task.isSuccessful)
        }
    }


}
```

Figure 29

The code above tests the User Login functionality of my application.

1. Firstly, I specify that I'm using JUnit4 as my test runner by using the @RunWith annotation.
2. Next I define a rule using the @get:Rule annotation. This indicates that the following code is a JUnit rule.
3. I then declare a variable and set it to an instance of the ActivityScenarioRule class which is rule defined by JUnit that allows the instrumented tests to run and interact with the Activity in a testing environment. I choose the WelcomeActivity class because this is where my User Login functionality exists within my application.

4. Finally, I write the actual test using the @Test annotation. The test first finds the sign in button on the Welcome Screen and performs a click. To verify this click has worked, I check that an appropriate layout element is displayed on the current screen. Then I create test variables for a test user in my database and attempt a Firebase Authentication login using these variables.
5. The assert(task.isSuccessful) method informs the system that the test has passed successfully.

The test results are displayed as shown below.

Figure 30

That was just one example of an instrumented test performed in my project. I will provide some more examples in the Appendices section below.

Below is a table of all the tests I have performed for my application based on the functional requirements stated above in the Requirement Specification section.

Number	Requirement	Tested	Passed	Comments
1	User Registration	**Yes**	**Yes**	User registration test has successfully passed as upon signing up to the system, the test user had been added to the database.
2	User Login	**Yes**	**Yes**	User login test has successfully passed as upon entering the test login credentials to the user login screen, the user has been transferred to the 'Home Screen'.
3	Personalised Unique QR Code	**Yes**	**Yes**	This test has successfully passed as upon a successful login, the generated QR code is displayed on the 'Home Screen'.
4	View Receipts	**Yes**	**Yes**	This test has successfully passed as upon a successful login, the accumulated receipts are displayed on the 'Receipt Screen'.
5	Accumulate Receipt	**Yes**	**Yes**	This test has successfully passed as upon scanning the QR code, a receipt has been added to the database and can be viewed from the 'Receipt Screen'.
6	Manage Receipt	**Yes**	**Yes**	This test has successfully passed us upon selecting the 'Delete Receipt Button', the selected receipt has been removed from the database and 'Receipt Screen'.

7	View Statistics	**Yes**	**Yes**	This test has successfully passed as upon navigation to the 'Statistics Screen' after receipt accumulation, a progress bar is displayed along with the number of receipts the user has accumulated.
8	Manage Users	**Yes**	**Yes**	This test has successfully passed as upon Admin Registration and Sign In, the Admin User can delete Users from the database.

9. Conclusions

To conclude, MeCeipt is an app for Android devices that allows Business and Customer Users to make a conscious decision to improve the environmental impacts physical receipt distribution has on the environment. I will discuss some advantages and disadvantages of the current state of the app.

Advantages	
1	MeCeipt demonstrated digital receipt transfer from business to user.
2	Provides a platform to store digital receipts.
3	Provides limited statistical information for users.
4	Allows businesses to sign up and reap the benefits of digital receipt distribution.

Disadantages	
1	In its current form, MeCeipt can only be used for demonstration purposes. It presents how the app would work in a real-life scenario if access to company's receipt data was granted.
2	Limited statistics may not be very beneficial but opens the door to more intricate statistical data.

10. Further Development or Research

There is so much potential involved with the future development of MeCeipt. Working in a real-life scenario, MeCeipt offers the potential for businesses and users to benefit greatly. For example, users could benefit from a rewards program implemented into MeCeipt which allows businesses to provide deals and discounts to users who use MeCeipt in their stores on a regular basis.

Other future developments could include the inclusion of different technologies such as NFC for a more seamless transaction experience.

One requirement that was intended for this project was the implementation of scanning physical receipts and storing them as digital receipts. This could be beneficial for users who already have physical receipts who would like to store them all in one place without the need to hold onto the physical copy.

11. References

[1] W. Hines, "Going Paperless: The Hidden Cost of a Receipt," Huffpost, 4 April 2013. [Online]. Available: https://www.huffpost.com/entry/going-paperless-the-hidde_b_3008587?ec_carp=4408988367690251169.

12. Appendices

12.1. Project Proposal

1. Objectives

The problem which my project will set out to solve is the environmental and economic impacts that receipt distribution has on retail companies. To solve these problems, this project will focus on the development of an app for Android devices which will provide digital receipt distribution to customers and allow them to view and manage all of the receipts they have accumulated. The app will allow businesses in the retail sector and customers to reap the benefits of not having to rely on paper receipts after a transaction. For businesses, such benefits would include:

- The reduction and/or elimination of receipt rolls.
- Reducing the costliness of receipt rolls.
- A reduction in paper and paper waste.
- A more environmentally friendly distribution of receipts to customers.

For customers, such benefits would include:

- Having a single place to store digital receipts.
- Eliminating the risk of misplacing receipts.
- Managing receipts in a user-friendly app.

2. Background

Having worked in one of the largest multinational retail companies in the world for the past 8 years, I've noticed massive change with relation to the adoption of digital technology to replace traditional paper files which include, but are not limited to rosters, reports, and payslips. These changes were made in an attempt to reduce the amount of paper waste produced by the company. However, one area I've noticed where the reduction of paper waste hasn't been given much attention to is the distribution of receipts to the customers.

Most retailers don't have an alternative method of distributing receipts to customers. One variation I've seen is email distribution, where the cashier will request the email address of a customer and a receipt will then be sent to them. However, this is not as seamless as it sounds as there may be language barriers or other such issues that may result in the misinterpretation of a customer's email address. I've yet to see a seamless, industry standard method of digital receipt distribution. I set out to resolve this issue with my project.

To resolve the problems I've laid out above and in the Objectives section above, I plan on creating an Android app that will interact with point-of-sale terminals and seamlessly transfer receipts from the retailer to the customer in a customer-friendly user-interface.

3. State of the Art

During my research I found applications with similar functionality to what I have in mind for my project. However, some of these apps require a paper receipt to be given to the customer regardless of the app existing or not. For example, one app I found, *Receipt Scanner*, required the customer to take a photo of a physical receipt which would then be digitally transformed and be accessible on the app. The aim of my app is to reduce the paper-waste and costliness of receipt paper by creating an app that will give both a business and a customer an option of an entirely digital receipt.

Other alternatives that already exist are email distribution of receipts, which I mentioned above. The issue I find with this method of receipt distribution is the potential misinterpretation of the customers' email address and the inefficiency of having to spell out the email address or type it out on an additional device. The idea of MeCeipt is to make the receipt distributing process seamless and fast for both the customer and retailer.

4. Technical Approach

For my project, I will be using a Waterfall Development Methodology, where I will first outline the requirements and scope of the project. This step will involve identifying the problem and outlining the requirements needed to solve it. This will be followed by the Design phase which will focus on the designing of a solution based on the requirements specified in the previous step. Once a design plan is in place, I will be able to begin coding my application. This will be followed the Testing phase which will aim to ensure the requirements are met to a satisfactory standard.

To identify the requirements, I will put myself in the role of a potential client and question myself on how such an app should best perform. I need to understand what my app should do and how it should work in a step-by-step manner. As soon as my requirements have been fully identified I can begin to convert the requirements into use cases and outline the tasks necessary to transform the requirements into functionality.

I will use an online Project Management software such as Monday or Click Up to create a project plan that I can easily follow and implement tasks for each use case I will have identified. Using a Project Management software will allow me to easily identify timelines and milestones so I can most efficiently complete tasks and have an organised development process for my project.

5. Technical Details

As of this point in my project, the technologies I will use for developing my application is Android Studio and Kotlin. Android Studio is a Development Environment specifically made for Android app development. Android Studio is a user-friendly and efficient IDE for Android app development with a built-in front-end design tool for Android Devices. Android Studio allows for seamless and efficient testing with its built-in Android device emulators which I will discuss further in the Testing section below.

Kotlin is the programming language I will use with Android Studio. Kotlin is an easy-to-learn language with the ability of cross-platform functionality and works seamlessly with popular object-oriented languages such as Java. This is an ideal selection of technology for my project as my app will require communication with an external software such as some point-of-sale application.

My database of choice will be Firebase as its Google's recommendation when building Android apps. Android Studio provides full integration of Firebase so it's the most logical choice for this project. Some further research may have to do be carried out as, although I intend for this app to be fully available to users regardless of internet connectivity, I need to find out how possible this is.

6. Special Resources Required

My application will require co-operation and communication with a point-of-sale terminal. At this stage of my project, however, I have yet to decide whether to develop my own point-of-sale terminal or to use an open source one.

I would also have to consider the potential of gathering a secondary dataset of retail purchases to test my application with but have not yet made the decision on whether to create this data myself or to gather it from the internet.

7. Project Plan

As I mentioned above in the Technical Approach section, I will be using the Waterfall Methodology during the development process for my project. The Waterfall Methodology splits the development process into different phases. These phases include:

- Requirements phase – This is the phase where I will take time to identify the requirements by understanding the problem this project seeks to solve. The requirements can then be morphed into Use Cases which will allow me to identify the problem more clearly.
- Design phase – The design phase will allow to create a solution to the problems or requirements defined in the Requirements phase.
- Code phase – This will likely be the lengthiest phase and will involve the coding of the application designed in the previous phase.
- Test phase – This phase will begin when the coding of the application is finished. It will involve ensuring that all requirements that were identified are met and functionality is of an adequate standard.

Below is a detailed structure of my project plan and all of the dates and deliverables included in it.

Project Plan		
Requirement Phase		
Task	**Task Description**	**Due Date**
Identifying Requirements	This task involves the gathering of functional and non-functional requirements.	7/11/22
Use Case Diagram	This task involves the creation of a high-level Use Case Diagram of MeCeipt.	10/11/22
Use Cases	This task involves the creation of Use Cases from the requirements identified.	11/11/22
Design Phase		
Task	**Task Description**	**Due Date**
Wireframes	This task involves the development of wireframe mock-ups of the screens within the system.	15/11/22
Sign-Up Screen Design	This task involves developing the front-end user interface for the sign-up screen of the application.	17/11/22

Sign-In Screen Design	This task involves developing the front-end user interface for the sign-in screen of the application.	18/11/22
Welcome Screen Design	This task involves developing the front-end user interface for the welcome screen of the application.	19/11/22
Home Screen Design	This task involves developing the front-end user interface for the home screen of the application.	20/11/22
Receipt Screen Design	This task involves developing the front-end user interface for the receipt screen of the application.	21/11/22
Statistics Screen Design	This task involves developing the front-end user interface for the statistics screen of the application.	22/11/22
Code Phase		
Task	**Task Description**	**Due Date**
User Registration	This task describes a user having the ability to register for the system.	1/12/22
Business User Registration	This task describes a business user having the ability to register for the system.	2/12/22
Sign-In Functionality	This task describes a user having the ability to sign into the system.	3/12/22
QR Code Generation	This task describes a QR code being generated upon user registration.	10/12/22
Mid-Point Presentation Submission	This task describes the deadline for the upload of Mid-Point Presentation.	20/12/22
Receive Receipt	This task involves a user receiving a digital receipt at a point-of-sale area.	1/3/23
View Receipts	This task involves a user having the ability to view the receipts they've accumulated.	3/3/23
Manage Receipts	This task involves a user having the ability to manage their receipts.	10/3/23
View Statistics	This task involves users having the ability to view their personalised statistics.	5/4/23
Test Phase		
Task	**Task Description**	**Due Date**
Testing User Registration	This task involves the testing of the user registration functionality.	4/1/23
Testing Business User Registration	This task involves the testing of the business user registration functionality.	5/1/23
Testing Sign-In	This task involves the testing of the user sign-in registration functionality.	6/1/23
Testing QR Code Generation	This task involves the testing of the QR code generation functionality.	12/1/23
Testing Receiving of Receipts	This task involves the testing of the receiving of receipts as a customer user.	1/4/23
Testing of Viewing Receipts	This task involves the testing of the viewing of receipts as a customer user.	10/4/23

Testing of Managing Receipts	This task involves the testing of the managing of receipts as a customer user.	20/4/23
Testing of Viewing Statistics	This task involves the testing of the viewing of statistics as a user.	5/5/23
14th MAY 2023 – SUBMISSION DEADLINE		

8. Testing

I plan on taking a test-driven development approach for my project meaning the requirements I define will be converted to test cases before being coded.

Android Studio has a very intuitive way of carrying out tests. Android Studio comes with an emulator for every version of Android and allows you to emulate many different Android powered phones. This allows you to test your application in real-time on any version of Android. However, this method of testing is quite resource heavy as your computer must be able to run your development environment (Android Studio) and the emulator. There is an alternative method that Android Studio supplies for testing which is connecting a physical Android device to your computer to test in real-time. Entering Developer Mode on any Android device and connecting it to Android Studio is probably the most efficient way to test and this is the method I will be using to test throughout the development process of my project.

I will document my testing my creating a test plan along with test cases and test scenarios. This approach will be carried out throughout the lifespan of my project and will be verifiable in my documentation for my final submission.

12.2. Reflective Journals

12.2.1. October 2022

Supervision & Reflection Template

Student Name	Shane Mulrooney
Student Number	x19454016
Course	BSHCSD4
Supervisor	Frances Sheridan

Month: October 2022

What? During October, I finalised the idea for my project and presented my pitch to my supervisor. I went into further details in my project proposal and went on to meet with my supervisor to iron out any uncertainties I had with my idea.
So What? Meeting with my supervisor to clear up the uncertainties I had was very beneficial and gives me a good foundation to begin work on my project. I still face the challenge of how to go about developing my app which will span over the course of the coming months.

Now What? There is much research I must do in order to begin working on my project which I plan to set out to do in the coming days and weeks.
Student Signature | Shane Mulrooney

12.2.2. November 2022

Supervision & Reflection Template

Student Name	Shane Mulrooney
Student Number	x19454016
Course	BSHCSD4
Supervisor	Frances Sheridan

Month: November 2022

What? In November, I created a wireframe mock-up for what I intend the front-end of my application to look like. I presented this to my supervisor to get feedback. I also presented functional requirements to my supervisor and have begun work on my documentation for my mid-point presentation.
So What? Creating a wireframe has given me a good foundation to begin designing the front-end of my application in Android Studio. The functional requirements allow me to create a proper plan and timeline based on the complexity of the requirements.
Now What? December is big month for the development of my project. My mid-point presentation is due, so I need to put a lot of attention into getting prepared for that.
Student Signature | Shane Mulrooney

12.2.3. December 2022

Supervision & Reflection Template

Student Name	Shane Mulrooney
Student Number	x19454016
Course	BSHCSD4

Supervisor	Frances Sheridan

Month: December 2022

What? The mid-point presentation for my project was scheduled for a date in December. For the presentation, I was required to have some sort of working prototype ready to be demonstrated. My objective was to have a prototype that included user authentication and functionality for logging in and out of the system. I also planned to demonstrate a unique QR code to be generated and displayed to each user.
So What? Developing the front-end user interface for the functionality I aimed to implement for the mid-point demonstration introduced me to Android Studio and using XML elements to structure screens for Android apps. Implementing user authentication introduced me to Firebase and Firestore as database solutions and Kotlin as the primary programming language for my project.
Now What? Unfortunately, I overestimated my time and ability I had to implement everything I aimed to for the mid-point presentation. I had successfully implemented and a sign-up and login/logout system with user authentication but failed to add the functionality for generating unique QR codes for users. This goal will be carried over to the month of January amongst other functionality such as a bottom navigation bar where users will be able to navigate between different screens within the application.

Student Signature	Shane Mulrooney

12.2.4. January 2023

Supervision & Reflection Template

Student Name	Shane Mulrooney
Student Number	x19454016
Course	BSHCSD4
Supervisor	Frances Sheridan

Month: January 2023

What? Due to my inability to complete the tasks I laid out for the mid-point presentation, I decided to ensure those parts of the functionality would be implemented by the end of January.
So What?

Implementing a unique QR code for each user has proved to be a more difficult task than I first realised. With the limited time I had during this month, I didn't implement this function, however, I did add the bottom navigation bar as I aimed to.
Now What? Having failed to implement some of the functionality laid out in my mid-point presentation I need to do further research on how to go about doing so. Another major piece of functionality that needs to be started is the interaction between my application and some point-of-sale terminal. I must find an appropriate open-source point-of-sale terminal to test this interaction and further develop my application to meet the requirements laid out in my Requirement Specification.

Student Signature	Shane Mulrooney

12.2.5. February 2023

Supervision & Reflection Template

Student Name	Shane Mulrooney
Student Number	x19454016
Course	BSHCSD4
Supervisor	Frances Sheridan

Month: February 2023

What? My aim for the end of February was to generate personalised QR codes for each user in the database and display it on the home screen when they log in. I have also decided to change the scope of the project and for demonstration purposes, abandon the involvement of a point-of-sale terminal. Instead, a business user signed in on a second device will handle the transfer of receipt data to the user. I also plan on using this month to catch up on testing for the functionality I have implemented.
So What? The generation of personalised QR codes proved to be a success. Upon signing in and navigating to the home screen, the QR code is now displayed to the user. Testing has been brought up to date with the latest functionality.
Now What? Having implemented the QR codes, my aim for next month is to implement the transfer of receipt data from the business user device to the customer user device. To verify this, I must implement the receipt screen where the users will view all of the receipts they have accumulated.

Student Signature	Shane Mulrooney

12.2.6. March 2023

Supervision & Reflection Template

Student Name	Shane Mulrooney
Student Number	x19454016
Course	BSHCSD4

Supervisor	Frances Sheridan

Month: March 2023

What? By the end of March, I aimed to have the view receipts functionality implemented. This is one of the most crucial requirements of the project and looked to be the most difficult to implement. I also aimed to have input validations implemented upon user registration.
So What? As expected, the view receipts functionality proved to be both difficult and time consuming, but it had been completed by the end of March along with input validations on the registration fields.
Now What? Now that what was expected to be the most difficult functionality has been implemented. I need to spend the month of April finalising the implementation and begin to focus on the final presentation. I need to implement admin functionality and the environmental statistics page which I don't predict to be the most time consuming. I also need to finish up on testing and make some front-end tweaks to result in a friendly UI experience.
Student Signature Shane Mulrooney

12.2.7. April 2023

Supervision & Reflection Template

Student Name	Shane Mulrooney
Student Number	x19454016
Course	BSHCSD4
Supervisor	Frances Sheridan

Month: April 2023

What? April is the last month that will involve the development of my app. It is crucial to have all of my requirements finalised and be ready to be presented. This includes finishing the remaining functionalities and adding finishing touches to the UI. Once completed I can then begin to finalise my documentation and presentation.
So What? I have completed the implementation of all of the functionalities laid out in my requirement specification. I added the finishing touches to the UI and the remaining functionalities including admin privileges and the statistics page.
Now What? Now that the implementation is completed, I can begin to finalise the documentation and start on my presentation and demonstration video due to be submitted on the 14th of May.

Student Signature	Shane Mulrooney

12.3. Testing

Here I will provide some further examples of Integration Testing I performed on my application.

```
@get:Rule
var signInActivityRule = ActivityScenarioRule(SignInActivity::class.java)

@Test
fun homeActivityTest() {
    val email = "test@email.com"
    val password = "123456"

    FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password).addOnCompleteListener() { task ->
        if (task.isSuccessful) {
            onView(withId(R.id.btnSignIn)).perform(click())
            onView(withId(R.layout.fragment_home)).check(matches(isDisplayed()))
        }
    }
}
```

Figure 31

Figure 31 shows an instrumented test to test that the Home Activity displayed as intended.

```
@get:Rule
var homeActivityRule = ActivityScenarioRule(HomeActivity::class.java)

@Test
fun homeNavigationTest() {
    onView(withId(R.id.qrCode)).check(matches(isDisplayed()))
    onView(withId(R.id.env)).perform(click())
    onView(withId(R.id.tfReceiptAmount)).check(matches(isDisplayed()))
    onView(withId(R.id.receipt)).perform(click())
    onView(withId(R.id.userReceiptRecyclerView)).check(matches(isDisplayed()))
    onView(withId(R.id.home)).perform(click())
    onView(withId(R.id.qrCode)).check(matches(isDisplayed()))

}
```

Figure 32

Figure 32 shows an instrumented test to test the functionality of the bottom navigation bar when a user is signed into the system.

```
@Test
fun viewReceiptsTest() {
    onView(withId(R.id.qrCode)).check(matches(isDisplayed()))
    onView(withId(R.id.receipt)).perform(click())
    onView(withId(R.id.userReceiptRecyclerView)).check(matches(isDisplayed()))
    onView(withText( text: "Test")).check(matches(isDisplayed()))
}
```

Figure 33

Figure 33 shows an instrumented test to test the functionality of the View Receipts requirement.

12.4. Project Plan

Project Plan			
Requirement Phase			
Task	**Task Description**	**Due Date**	**Completed**
Identifying Requirements	This task involves the gathering of functional and non-functional requirements.	7/11/22	**Yes**
Use Case Diagram	This task involves the creation of a high-level Use Case Diagram of MeCeipt.	10/11/22	**Yes**
Use Cases	This task involves the creation of Use Cases from the requirements identified.	11/11/22	**Yes**
Design Phase			
Task	**Task Description**	**Due Date**	**Completed**
Wireframes	This task involves the development of wireframe mock-ups of the screens within the system.	15/11/22	**Yes**

Sign-Up Screen Design	This task involves developing the front-end user interface for the sign-up screen of the application.	17/11/22	Yes
Sign-In Screen Design	This task involves developing the front-end user interface for the sign-in screen of the application.	18/11/22	Yes
Welcome Screen Design	This task involves developing the front-end user interface for the welcome screen of the application.	19/11/22	Yes
Home Screen Design	This task involves developing the front-end user interface for the home screen of the application.	20/11/22	Yes
Receipt Screen Design	This task involves developing the front-end user interface for the receipt screen of the application.	21/11/22	Yes
Statistics Screen Design	This task involves developing the front-end user interface for the statistics screen of the application.	22/11/22	Yes
Code Phase			
Task	**Task Description**	**Due Date**	**Completed**
User Registration	This task describes a user having the ability to register for the system.	1/12/22	Yes
Business User Registration	This task describes a business user having the ability to register for the system.	2/12/22	Yes
Sign-In Functionality	This task describes a user having the ability to sign into the system.	3/12/22	Yes
QR Code Generation	This task describes a QR code being generated upon user registration.	10/12/22	Yes
Mid-Point Presentation Submission	This task describes the deadline for the upload of Mid-Point Presentation.	20/12/22	Yes
Receive Receipt	This task involves a user receiving a digital receipt at a point-of-sale area.	1/3/23	Yes
View Receipts	This task involves a user having the ability to view the receipts they've accumulated.	3/3/23	Yes
Manage Receipts	This task involves a user having the ability to manage their receipts.	10/3/23	Yes
View Statistics	This task involves users having the ability to view their personalised statistics.	5/4/23	Yes
Test Phase			
Task	**Task Description**	**Due Date**	**Completed**
Testing User Registration	This task involves the testing of the user registration functionality.	4/1/23	Yes
Testing Business User Registration	This task involves the testing of the business user registration functionality.	5/1/23	Yes
Testing Sign-In	This task involves the testing of the user sign-in registration functionality.	6/1/23	Yes
Testing QR Code Generation	This task involves the testing of the QR code generation functionality.	12/1/23	Yes

Testing Receiving of Receipts	This task involves the testing of the receiving of receipts as a customer user.	1/4/23	**Yes**
Testing of Viewing Receipts	This task involves the testing of the viewing of receipts as a customer user.	10/4/23	**Yes**
Testing of Managing Receipts	This task involves the testing of the managing of receipts as a customer user.	20/4/23	**Yes**
Testing of Viewing Statistics	This task involves the testing of the viewing of statistics as a user.	5/5/23	**Yes**
14th MAY 2023 – SUBMISSION DEADLINE			

12.5. GitHub & Final Presentation Links

Final Presentation: https://youtu.be/ZIw9OaqZo78

GitHub: https://github.com/shanem1996/MeCeipt