A Critical Analysis of the Project Development

APP : Combine Voucher System Among Multiple Irish Stores

Author *:* *Paula Dwan* *(paula.dwan@gmail.com)*

FAO : *Dominic Carr* *(dominic.carr@ucdconnect.ie)*
Tadhg O'Sullivan *(t.osullivan@ucd.ie)*
Barnard Kroon *(barnard.kroon@ucdconnect.ie)*

Course : *MSc Advanced Software Engineering*

Module : *COMP–47330 : Practical Android Programming*

Student ID *:* *13208660*

Due Date *:* *01-May-2015*

TABLE OF CONTENTS

INDEX OF TABLES

1 OVERVIEW OF APPLICATION

1.1 HIGH-LEVEL FUNCTIONALITY

App based rewards / points / vouchers scheme for customers of national and/or local stores.

- Customer registers using Android (or web-page login);
- Customers details are saved in central database;
- Customer enters a new store and scans QR code to sign-up (or web-page);
- Customer receives all vouchers via app – mailings, points, money-off, mailings, event vouchers, etc.;
- Customer can login with Facebook;
- Customer perhaps also receives proximity vouchers

1.2 APP FLOW – NEW USER (START TO END)

This flow covers a new user. The detailed start-to-end flow covers the user experience from when he/she starts the app to leave/closing it. A new user has some of the same views as an existing user :

- Home / Welcome

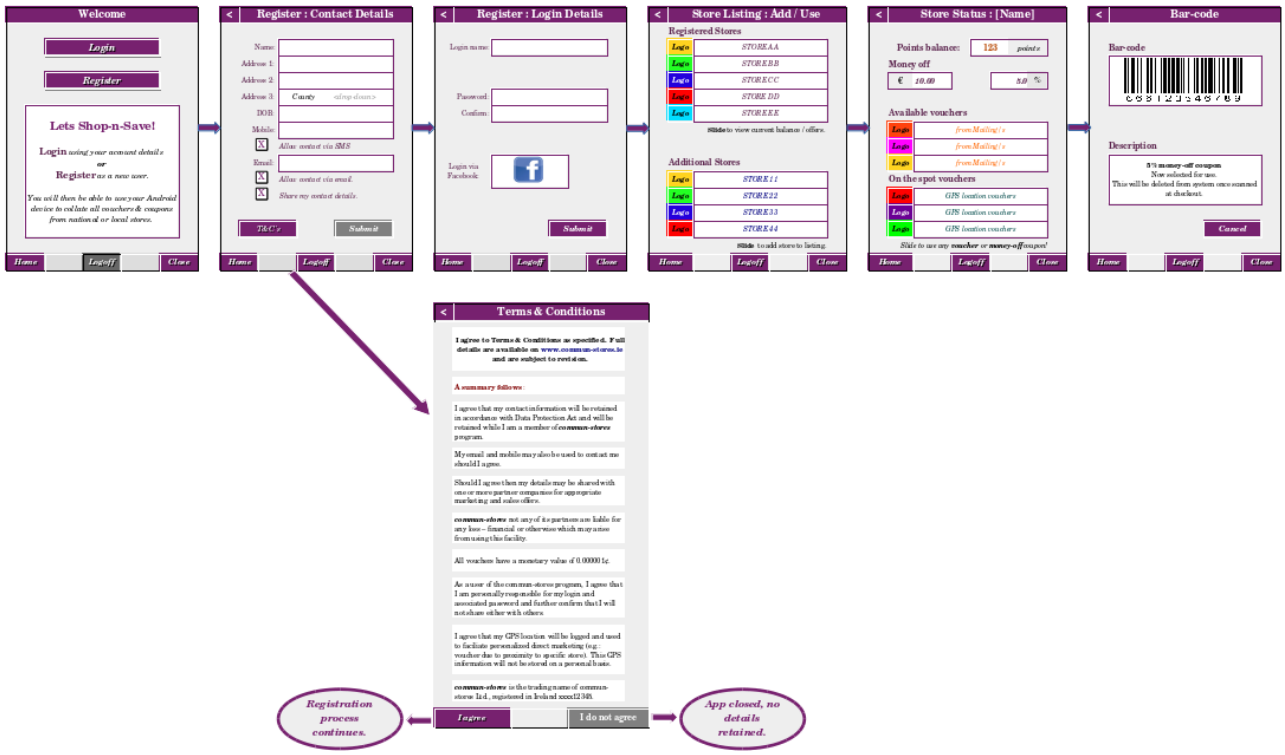
Once registered, the following are available :

- Store Listing : Add / Use | Store Status [store_name] | Bar-code / QR-code | Cancel Registration

However, he/she also has views restricted to new users only :

- Terms & Conditions | Register : Contact Details | Register : Login Details

1.2.1 MOCK-UP : START-TO-END FLOW



1.2.2 ACTUAL : START-TO-END FLOW

<what was finally delivered – multiple if needed>

1.3 APP FLOW – EXISTING USER (START TO END)

This flow covers an existing user. The detailed start-to-end flow covers the user experience from when he/she starts the app to leave/closing it. An existing user has some of the same views as a new user :

- Home / Welcome

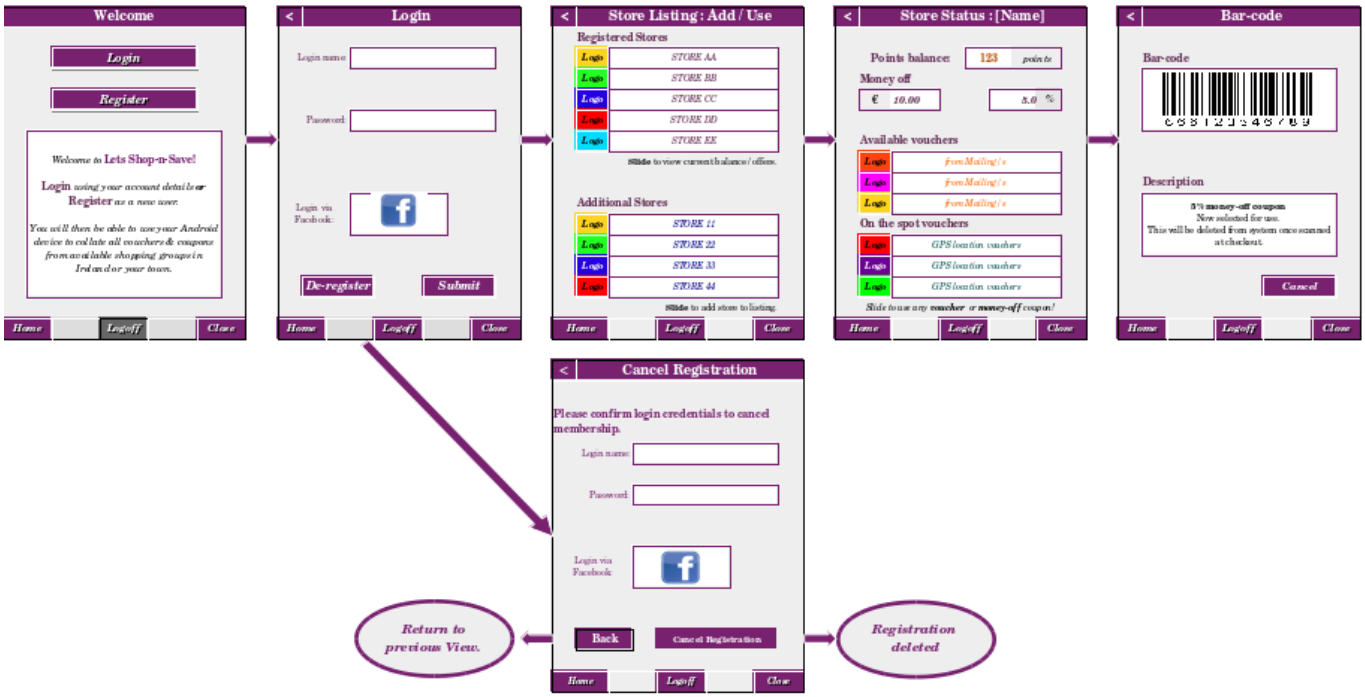
Once signed in the following are available :

- Store Listing : Add / Use | Store Status [store_name] | Bar-code / QR-code | Cancel Registration

However, he/she also has views restricted to existing users only :

- Login

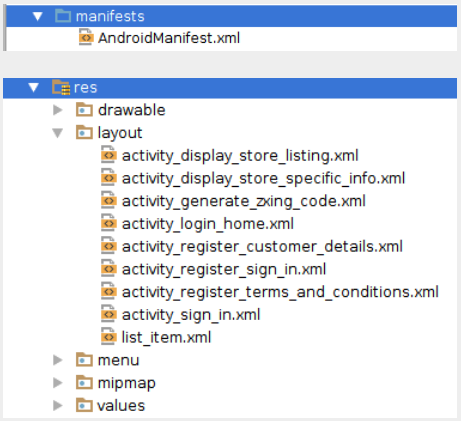
1.3.1 MOCK-UP : START-TO-END FLOW



1.3.2 ACTUAL : START-TO-END FLOW

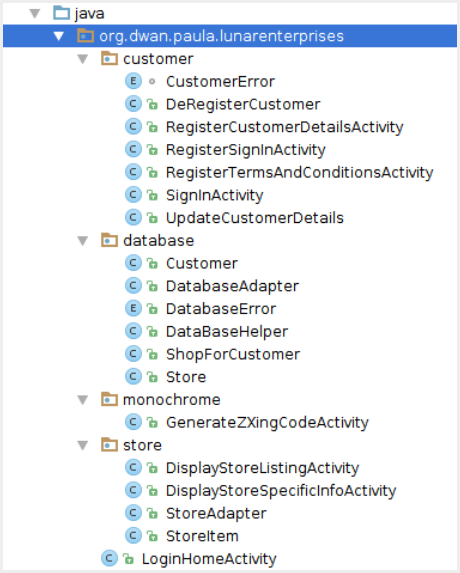
<what was finally delivered – multiple if needed>

1.4 APP FILE STRUCTURE – ANDROID STUDIO



Manifest file → specifics activity flow : parent → child.

- /drawable → graphic files, e.g.: app and store logos.
- /layout → Specific layout for each activity.
- /menu → any menu specific files, e.g.: activity bar layout.
- /values → resource files for the app, e.g.: strings.xml.



File and directory structure as implemented in Android Studio.

- org.dwan.paula.lunarenterprises → main project is which contains the Home Activity.
- org.dwan.paula.lunarenterprises.customer → all customer specific activities.
- org.dwan.paula.lunarenterprises.database → all database specific activities as well as helper and adapter.
- org.dwan.paula.lunarenterprises.monochrome → QR-Code / Bar-code generation activities.
- org.dwan.paula.lunarenterprises.store → all store specific activities.

Table 1:1 – Directory Structure (high-level overview of project structure as implemented)

1.5 App Flow – Changes to Common Implementation

This section details any common changes with the app flow or functionality.

1.5.1 Reasons Why?

Remove : *Bottom bar*	I removed the bottom bar for `[Home], [Logoff]` and `[Close].` I instead added an Overflow menu in the Action Bar, thus I then had the same functionality on each page and just checked that the customer was signed in (otherwise could not [Logoff]). I also changed the action name from [Logoff] to [Sign Out] to better match the initial option of [Sign In].
Remove : *Cancel Registration*	This activity was removed and replaced with an option in the Action Bar. This functionality was added to the Overflow menu present in every activity once the customer signs in or registers. Thus this functionality was easily accessible all the time and not just at one point in the process. The user is asked to confirm his/her password as additional confirmation.
Remove : *Facebook Login*	I did not have time to implement this. If this app is developed further (if some shops are interested) then it would be a good feature to have as the customer would not need to remember another login name and password. It would require updates to the database to validate live against Facebook as it would not be ideal to save the password and user name locally or in SQLite (the customer could update directly in Facebook or delete his/her Facebook account).
Add : *Color Schemes*	**[New User]** → **[Existing User]** → **[Shops]** This ensured that each customer interaction was easily recognized. Different color implies different functionality and requirements on the customer and app.

1.5.2 Advantages of Changes Made

Remove : *Bottom bar*	Better use of screen space and the bottom of each screen was now available for the main layout. This also resulted in a cleaner display which is more in keeping with today's apps.
Remove : *Cancel Registration*	Removal of one activity – reduce app size and also simplify usage for the customer, just click the drop-down option to '*Yes! I want to cancel my registration.*'
Remove : *Facebook Login*	None really as it is reduced functionality. However it does simplify the app for the first iteration.
Add : *Color Schemes*	Improved appearance and customer should find it easier to recognize where he/she is when using the app.

1.5.3 Disadvantages of Changes Made

Remove : *Bottom bar*	It is always possible to split the Action Bar when a smaller screen size applies.
Remove : *Cancel Registration*	None really.
Remove : *Facebook Login*	Reduced functionality.
Add : *Color Schemes*	Meant that rounded cell shapes had to be duplicated for each color scheme – taking more time to complete and also that I had to ensure the correct one was used in the appropriate activity / situation.

1.6 App Usage – Devices & Orientations / Layouts

Phone – typical	I restricted orientation to portrait as landscape does not really make sense when completing a form. The QR-Code was also generated in portrait layout while the Bar-code was generated as landscape only. The orientation reverted to portrait once the user returned to the previous activity or to home via the Activity Bar.
Tablet	The only difference between a phone and Tablet implementation would be the QR-Code / Bar-code placed to the right-hand side of the store listing. Registration of customer details activity should be retained as a separate activity, thus retaining registration of sign-in details (name and password) as separate activities. In future, more customer information may be requested at a later stage on the registration sign-in without impacting current design and also perhaps training videos on usage or an introductory offer from the store the customer is located in could be added to the right-hand side of the Register SignIn Activity later on. Currently only average Android phones (mid-level → Samsung S4, Samsung Galaxy Note) as few shoppers would take a tablet with them while shopping. The most viable device would be S4 or similar in size, basically a regular android device.
Phone – small	Not covered as after checking on ZXing and displays on the small phones, the screen is too small for the bar-code or the QR-code to be legible for the scanners. Depending on the number of older devices in user and by whom (core customer group?), it may be required to send the bar-code using digits only by SMS or email. However, this would need to be looked at as the shop assistant would need to type in the code and this could slow down sales line at check-outs.

2 PROJECT RUN-THROUGH

2.1 ACTIVITY – WELCOME

2.1.1 WHAT THIS ACTIVITY DOES?

Home activity – this is first layout of the app where an existing user signs back in or a new user begins the registration process.

2.1.2 VIEW : MOCK-UP -V- ACTUAL

Mock-up	Actual
	

Table 2:1 – Lunar Store initial screen → Welcome screen

Difference/s	Common updates (see : App Flow – Common Discrepancies) and the following : 1. The buttons were relocated to underneath the introductory text. 2. The name of the app was changed.
Reason/s	1. The user will read from top to bottom so it makes more sense to have buttons underneath the introduction. 2. No real reason just want a moon for an app logo.

2.1.3 CODE : LOGINHOMEACTIVITY.JAVA

LoginHomeActivity.java contains methods called for onClick listener for each button :

[Register]

Opens ***RegisterTermsAndConditionsActivity***

Opens the T&C activity → once the new user agrees to the T&Cs as proposed, he/she can then register personal details and options in a new activity, followed by user name (unique) and password in a subsequent activity.

This listener, if implemented, calls the method :

```
public void registerNewUser(View view) {
    Log.d(CLASS_NAME, "\t: register as new user, call new layout ...");
    Intent intentRegister = new Intent(getApplicationContext(),
     RegisterTermsAndConditionsActivity.class);
    startActivity(intentRegister);
}
```

[Sign In]	Opens ***SignInActivity*** An existing user opens the sign-in activity and logs in using an existing user name and corresponding password. This listener, if implemented, calls the method signInExistingUser, which calls the class SignInActivity.java with the intent intentSignin.

2.1.4 LAYOUT : ACTIVITY_LOGIN_HOME.XML

A <LinearLayout> was used for the overall display and all informational text strings were displayed using <TextViews>'s. In addition, there are two <Button>'s and each has an OnClick listener :

[Register]	was formatted using *drawable/cell_round_navy.xml,* and had an OnClick listener to call the RegisterContactDetails activity
[Sign In]	*was formatted using drawable/cell_round_purple.xml,* and had an OnClick listener to call the SignIn activity

2.2 *ACTIVITY – TERMS & CONDITIONS (NEW USER)*

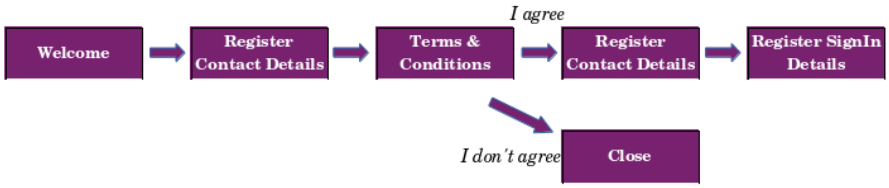
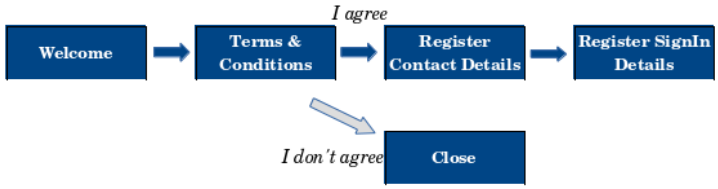
2.2.1 WHAT THIS ACTIVITY DOES?

Displays a summary of the Terms & Conditions to the potential customer. The user is asked to confirm that he/she has read and also understood the T&C's before the **[I Do Agree]** button is enabled. When clicked this button causes the RegisterContactDetails activity to open. The **[I Don't Agree]** button is always enabled and closes the app. As the user has not entered any details, there is no need to clear any information from the database or from the app.

2.2.2 VIEW : MOCK-UP -V- ACTUAL

Mock-up	Actual
	

Table 2:2 – Register New Customer → Agree to T&Cs

Difference/s	Common updates (see : App Flow – Common Discrepancies) also the flow was changed.  From :  To :
Reason/s	The user goes immediately to T&Cs and can decide if Lunar Stores is a viable app to use. Also, no details need to be stored and thus deleted if the user chooses **[I Don't Agree]** button. Until the user clicks on 'I confirm that I have read and understood the T&Cs', the **[I Do Agree]** button is disabled. Once clicked, RegisterCustomerDetails activity opens.

2.2.3 CODE : REGISTERTERMSANDCONDITIONSACTIVITY.JAVA

LoginHomeActivity.java contains an onClickListener for each button :

[I Do Agree] Opens RegisterContactDetailsActivity

This listener, if implemented, calls the method AgreeToTermsAndConditions which basically opens the activity to register new customer activity.

```
public void AgreeToTermsAndConditions(View view) {
    Log.d(CLASS_NAME, "\t: Agree to T&C's, call new layout ...");
    Intent intentRegisterDetails = new Intent(getApplicationContext(),
        RegisterCustomerDetailsActivity.class);
    startActivity(intentRegisterDetails);
}
```

[I Don't Agree] Closes the app → basically calls finish() and closes database if open.

```
public void DoNotAgreeToTermsAndConditions(View view) {
    Log.d(CLASS_NAME, "\t: Do Not Agree to T&C's, finish ...");
    finish();
}
```

2.2.4 LAYOUT : ACTIVITY_REGISTER_TERMS_AND_CONDITIONS.XML

A <TableLayout> was used for the overall display. This was embedded in a <ScrollView> ensuring that any items off screen initially could be scrolled to by the user.

In order to fake the bullets in the left cell using *drawable/cell_round_nay_bullet.xml.* All T&C text strings were displayed using <TextViews>, and Multiple Line display was enabled in case needed – depending on device.

```
<TableRow ... ... ... >
    <TextView
        android:id="@+id/textviewTandCBullet01"
        android:background="@drawable/cell_round_navy_bullet"
        android:gravity="center_horizontal"
        android:text="@string/tvTandCsBullet"
        android:textColor="@color/grey_pale"
        android:textStyle="bold"></TextView>
    <TextView
        android:id="@+id/textviewTandC01"
        android:layout_weight="1"
        android:ems="5"
        android:maxLines="10"
        android:padding="5dp"
        android:singleLine="false"
        android:text="@string/tvTermConditions01"
        android:textAppearance="@android:style/TextAppearance.Small"
        android:textStyle="italic"></TextView>
</TableRow>
```

Where no bullets applied then the string spanned two cells, with a null spacer in the first cell.

```
<TableRow ... ... ... >
    <TextView android:id="@+id/nullStr02"></TextView>
    <TextView
        android:id="@+id/textviewTandCSummary"
        android:layout_span="2"
        android:layout_weight="1"
        android:background="@color/grey_pale"
        android:maxLines="5"
        android:padding="5dp"
        android:singleLine="false"
        android:text="@string/tvTermConditionsIntro02"
        android:textColor="@color/blue_dark"
        android:textStyle="bold"></TextView>
</TableRow>
```

There are two <Button>'s and each has an OnClick listener :

[I Do Agree]	was formatted using *drawable/cell_round_navy.xml,*
[I Don't Agree]	was formatted using *drawable/cell_round_grey.xml.*

2.3 ACTIVITY – REGISTER CUSTOMER CONTACT DETAILS (NEW USER)

2.3.1 WHAT THIS ACTIVITY DOES?

Once the user enters contact information and checked (or not) SMS, Email or 3rd party contact then he/she may click on Submit, the data is retained and the new customer is then in this activity, is asked to provide a user name and password.

Rather than detailed help functionality – separate ReadMe or help activity – I used hints for each <EditText> field.

2.3.2 VIEW : MOCK-UP -V- ACTUAL

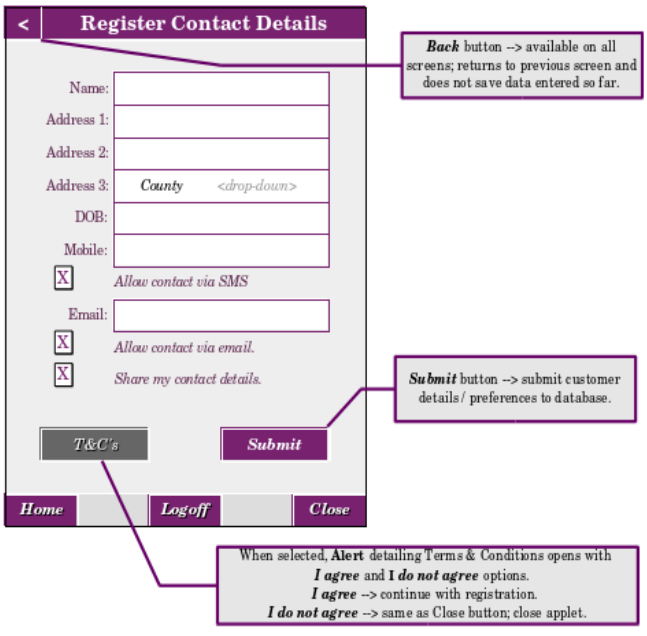
Mock-up	Actual
	

Table 2:3 – Register Customer New Customer → Contact details with blank details

Table 2:4 – Register New Customer → Contact details with fields and check boxes completed

Difference/s	Common updates (see : App Flow – Common Discrepancies) also the flow was changed. 1. It was initially thought to have all <TextEdit> and <CheckBox> disabled until the T&C's had been agreed to. This was changed to a flow revision, which is explained in section View : Mock-up -v- Actual of Activity – Terms & Conditions (New User). 2. I also removed the DOB requirement as it is possible to lie online and claim to be younger or older than he/she actually is. Thus, I decided to remove this data request. Instead, as always occurs the sales assistant will verify the age of the customer and if the offer is not valid for the customer's age then the sales assistant my refuse service in this instance.
Reason/s	1. Improve flow by moving T&C's and also improved functionality / user experience by implementing common updates. 2. Age as I mentioned age may be faked, so up to sales assistant to validate age appropriate sales (e.g.: alcohol, cigarettes.)

2.3.3 CODE : REGISTERCONTACTDETAILSACTIVITY.JAVA

[Submit] The customer details as entered are validated and if okay are added to an intent which is sent to a new activity where he/she provides user name (unique) and password.

```
public void SubmitCustomerContactDetails(View view) {
    Log.d(CLASS_NAME, "\t: accept and save customer details, call new
        layout ...");
    Intent intentRegisterDetails = new Intent(getApplicationContext(),
        RegisterSignInActivity.class);
    intentRegisterDetails.putExtra(CUST_NAME1, etName1.toString());
    intentRegisterDetails.putExtra(CUST_NAME2, etName2.toString());
    intentRegisterDetails.putExtra(CUST_ADDRESS1, etAddress1.toString());
    intentRegisterDetails.putExtra(CUST_ADDRESS2, etAddress2.toString());
    intentRegisterDetails.putExtra(CUST_ADDRESS3, etAddress3.toString());
    intentRegisterDetails.putExtra(CUST_ADDRESS4, etAddress4.toString());
    intentRegisterDetails.putExtra(CUST_EMAIL, etEmail.toString());
    intentRegisterDetails.putExtra(CUST_MOBILE, etMobile.toString());
    intentRegisterDetails.putExtra(CUST_EMAIL_OKAY, emailOkay.toString());
    intentRegisterDetails.putExtra(CUST_MOBILE_OKAY, smsOkay.toString());
    intentRegisterDetails.putExtra(CUST_3PARTY_OKAY, thirdPartyOkay.toString());
    startActivity(intentRegisterDetails);
}
```

[Clear All] All <EditText> strings as entered, are cleared, for example :

```
etMobile.getText().clear();
```

All <CheckBox> as clicked, are cleared, for example :

```
chkAllowSMS.setChecked(false);
```

Values of <CheckBox> for SQLite database are also reset to 0 → false, for example :

```
smsOkay = "0";
```

There are also private validation methods; e.g.: RegEx of strings entered, Email is valid format using android.util.Patterns, if allow contact by email is clicked that an email is provided; which are used by validateActivityFields(). Validation is carried out in the ***validateActivityFields*** method and once completed and passed, the Submit button is enabled.

The values for the <EditText> fields and the equivalent boolean values for SMS, Email and 3rd Party contact allowed (TRUE = '1' | FALSE = '0') are saved for later addition to the database.

As previously, the Action Bar was created and as the user has not yet registered, the <De-Register> option is not available.

2.3.4 LAYOUT : ACTIVITY_REGISTER_CONTACT_DETAILS.XML

Again, I used a <TableLayout> embedded in a <ScrollView>with column 1 used for titles suc as <Name>, <Address>, <Mobile>, and <Email>. These were all <TextField> formatted using *drawable/cell_round_navy.xml.* Column 2 contained the <EditText> fields which were used to retrieve the customers contact details. Column 2 was also used for the <CheckBox> options permitting contact via SMS, Email or from 3rd parties. Two buttons applied :

[Submit]	was formatted using *drawable/cell_round_navy.xml,* and had an onClick listener assigned.
[Clear All]	was formatted using *drawable/cell_round_grey.xml,* and had an onClick listener assigned.

2.4 ACTIVITY – REGISTER SIGNIN DETAILS (NEW USER)

2.4.1 WHAT THIS ACTIVITY DOES?

The new customer provides a user name and password, which is confirmed to ensure the user knows what he/she typed. The original password and the confirmation password are checked if equal, if not the user is asked to re-input them. The customer name is checked that it is lower case. Finally, both user name and login are checked for uniqueness. Once all validations are completed, the user contact details as well as user name and password are added to the SQLite database.

2.4.2 VIEW : MOCK-UP -V- ACTUAL

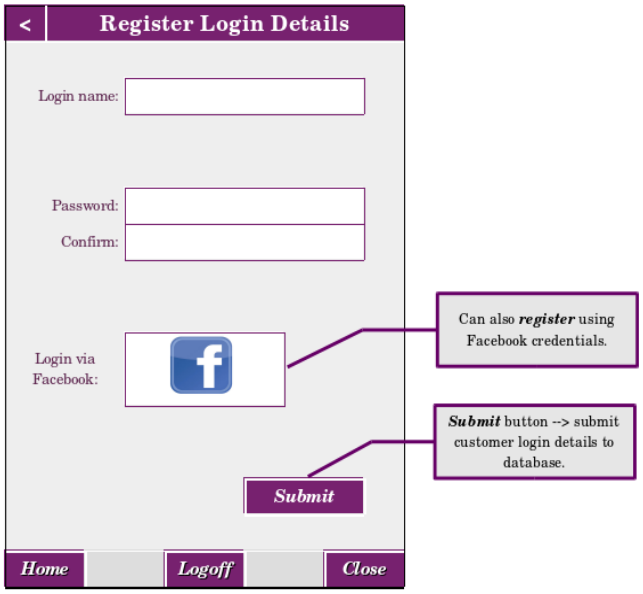
Mock-up	Actual
	

Table 2:5 – Register New Customer → Add login details

Difference/s	Common updates (see : App Flow – Common Discrepancies).
Reason/s	N/A

2.4.3 CODE : REGISTERSIGNINDETAILSACTIVITY.JAVA

In the previous activity the user provided his contact details which were validated for not Null, length and adhering to RegEx formatting (email, phone, etc.). No details were validated against the database. A customer may want the option of opening a second account, thus duplication of contact details is not relevant. However, sign-in details must be unique, so the user name must not exist in the database already. Duplicate passwords for different may occur and cannot be prevented. If ten users want to use *'password'* then that is fine but the user login name must be unique. **[Submit]** button is disabled until all fields are validated.

[Submit] The customer user name and password as entered are validated and if okay are added the SQLite database for that customer, all previously entered details are also added to the customer.tb table. The store listing activity now opens and the customer may start using the app. New customer created :

```
private void populateCustomerDetailsFields(Intent intentRegister) {
   Log.d(CLASS_NAME, "\t: populate EditFields using customer login details as
      entered ...");
   etNewUser = (EditText) findViewById(R.id.edittextRegisterUser);
   etNewPassword = (EditText) findViewById(R.id.edittextPasswordRegister);
   if (intentRegister != null) {
      customer.setName1(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_NAME1));
      customer.setName2(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_NAME2));
      customer.setAddress1(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_ADDRESS1));
      customer.setAddress2(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_ADDRESS2));
      customer.setAddress3(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_ADDRESS3));
      customer.setAddress4(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_ADDRESS4));
      customer.setMobile(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_MOBILE));
      customer.setEmail(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_EMAIL));
      customer.setEmailOk(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_EMAIL_OKAY));
      customer.setMobileOk(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_MOBILE_OKAY));
      customer.setThirdPartyOk(intentRegister.getStringExtra
         (RegisterCustomerDetailsActivity.CUST_3PARTY_OKAY));
   }
   btnRegisterSubmit = (Button) findViewById(R.id.buttonSignInSubmit);
   btnRegisterSubmit.setEnabled(false);
   if (validateAnyEditFieldUsed(new EditText[]{etNewUser, etNewPassword})) {
      if (validateActivityFields()) {
         customer.setNameLogin(etNewUser.getText().toString());
         customer.setPasswordLogin(etNewPassword.getText().toString());
         btnRegisterSubmit.setEnabled(true);
      }
   }
}
```

Added to customer.tb in database :

```
public void registerCustomerSQLDetails(View view) {
   Log.d(CLASS_NAME, "\t: validate & register new customer in SQL database ...");
   databaseAdapter = new DatabaseAdapter(this);
   try {
      databaseAdapter.open();
   } catch (SQLException e) {
      Log.d(CLASS_NAME, "\t: onCreate - register new user - SQL exception...");
      e.printStackTrace();
   }
   databaseAdapter.createCustomer(customer);
}
```

[Clear All] User name and password (Enter and confirm) are reset to the defaults, for example :

```
etNewUser.getText().clear();
etNewPassword.getText().clear();
etConfirmNewPassword.getText().clear();
```

2.4.4 LAYOUT : ACTIVITY_REGISTER_SIGN_IN_DETAILS.XML

Again, I used a <TableLayout> with column 1 used for titles <User Name> and <Password>. These were all <TextField> formatted using *drawable/cell_round_navy.xml.* The <TableLayout> was enclosed in a <ScrollView> in case needed. Column 2 contained the <EditText> fields which were used to retrieve the customers user name, initial password and the same password as confirmation. As previously, there were two buttons :

[Submit]	was formatted using *drawable/cell_round_navy.xml,* and had an onClick listener assigned.
[Clear All]	was formatted using *drawable/cell_round_grey.xml,* and had an onClick listener assigned.

2.5 ACTIVITY – SIGN IN (EXISTING USER)

2.5.1 WHAT THIS ACTIVITY DOES?

The existing user signs in using an existing user name and password. The Store Listing activity then opens with existing information and points saved to current date.

2.5.2 VIEW : MOCK-UP -V- ACTUAL

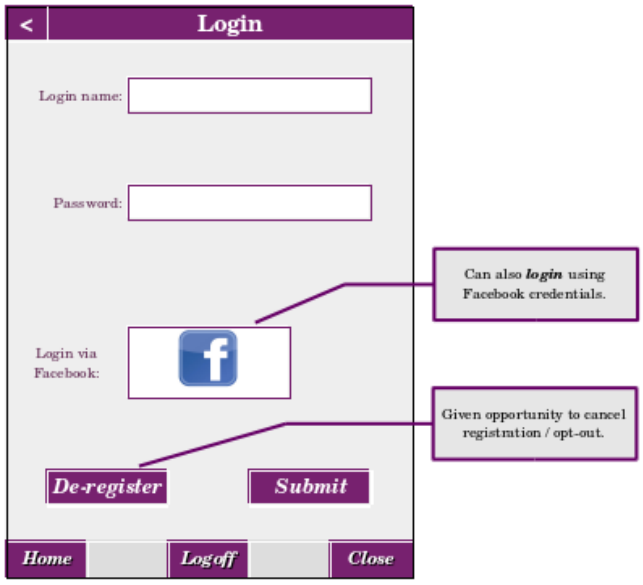
Mock-up	Actual
	

Table 2:6 – Existing User SignIn

Difference/s	Common updates (see : App Flow – Common Discrepancies) and De-register was moved to the common Action Bar. Also as in Register Customer Login Details, a Clear All button was added to reset Login Name and Password to default values.
Reason/s	Improved functionality and ease of customer use.

2.5.3 CODE : SIGNINACTIVITY.JAVA

The user provides his/her sign-in name and associated password which are validated for not Null, and against the database. Again validation is completed once the Submit button is clicked.

[Submit] The customer user name and password as entered are validated against the SQLite database for that customer. If valid then the store listing activity now opens and the customer may start using the app.

```
public void submitCustomerSQLDetails(View view) {
    Log.d(CLASS_NAME, "\t: validate sign-in details as entered by existing
        customer & open next activity...");

    // add custId to global variable
    Customer customer = new Customer();
    final GlobalVariableCustomerId globalVariableCustomerId =
        (GlobalVariableCustomerId) getApplicationContext();
    customer =  databaseAdapter.getCustomerUsingUserName
        (etUser.getText().toString());
    globalVariableCustomerId.setCustId(customer.getId());
    // send off intent
    Intent intentSignInDetails = new Intent(getApplicationContext(),
        DisplayStoreListingActivity.class);
    startActivity(intentSignInDetails);
}
```

[Clear All] User name and password are reset to the defaults, for example :

```
etLoginUser.getText().clear();
etLoginPassword.getText().clear();
```

2.5.4 LAYOUT : ACTIVITY_SIGN_IN.XML

Again, I used a <TableLayout> with column 1 used for titles <User Name> and <Password>. These were all <TextField> formatted using *drawable/cell_round_purple.xml.* The <TableLayout> was enclosed in a <ScrollView> in case needed. Column 2 contained the <EditText> fields which were used to retrieve the customers name and password. As previously, two buttons were used :

[Submit]	was formatted using *drawable/cell_round_purple.xml,* and had an onClick listener assigned.
[Clear All]	was formatted using *drawable/cell_round_grey.xml,* and had an onClick listener assigned.

2.6 ACTIVITY – CANCEL REGISTRATION (EXISTING USER)

2.6.1 VIEW : MOCK-UP -V- ACTUAL

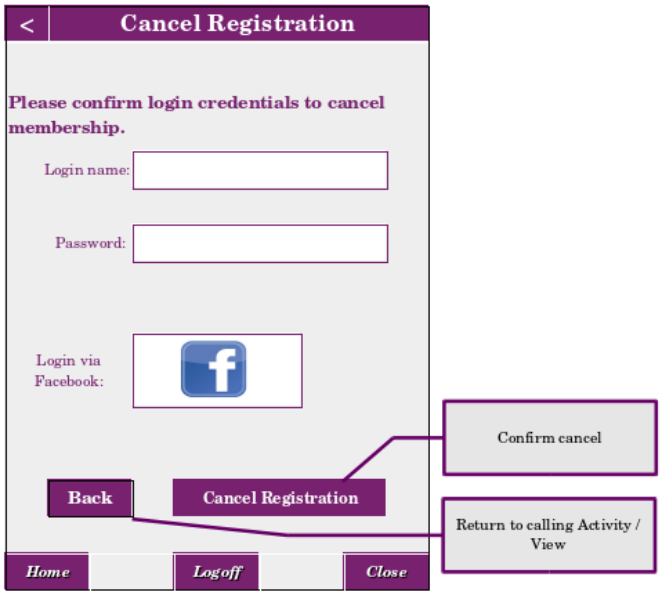
Mock-up	Actual
	

Table 2:7 – Cancel Registration Activity → replaced with ActionBar OverFlow menu option.

Difference/s	Deleted activity altogether, used an option in the ActivityBar overflow menu. A simple solution and this also provides access to the to the De-register option in all valid activities. Obviously, the use may not de-register until he/she has actually registered to use the app so this menu option is not available until then.
Reason/s	The user has already logged in, so it makes more sense to just use an option in the Activity Bar drop down and request the use to confirm his/her request to cancel registration.

2.7 ACTIVITY – STORE LISTING : ADD / USE (NEW & EXISTING USERS)

2.7.1 WHAT THIS ACTIVITY DOES?

Provides a listing **(i)** stores the customer collects points from currently and listing **(ii)** additional stores available. Customer then selects a store from listing **(i)** to see current balance and use points if applicable and if wished, and from listing **(ii)** to add that store to the list of stores he/she collects points from.

2.7.2 VIEW : MOCK-UP -V- ACTUAL

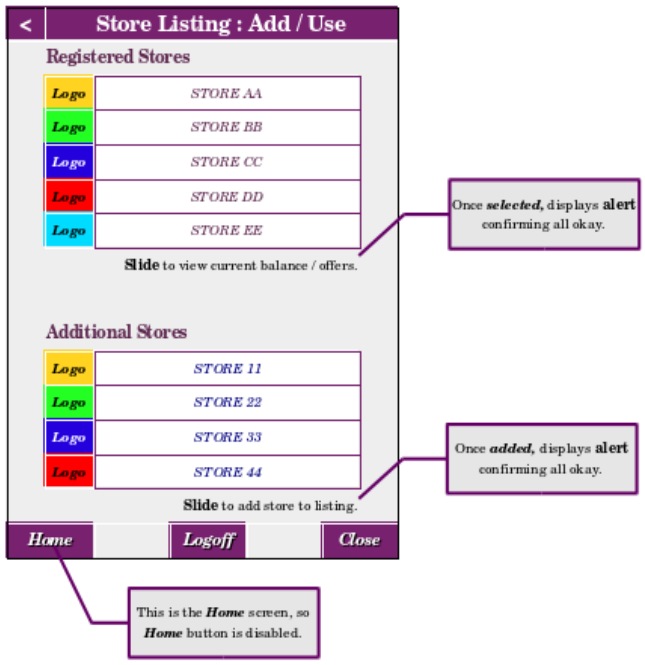
Mock-up	Actual
	

Table 2:8 – Store Listing → those in use and additional ones available.

Difference/s	Common updates (see : App Flow – Common Discrepancies). 1. Slide was not enabled, click on the list item was instead. 2. Also the store listing was combined to *<Register Stores>* and *<Additional Stores>*.
Reason/s	1. Easier for user to note what item he/she selects and thus less likelihood of error on smaller screens. 2. Easier to read and the option to add is now a button → only valid for stores

2.7.3 CODE : DISPLAYSTORELISTING.JAVA

Once the activity opens, populateListViewStores is called by onCreate. This creates a listing of all stores available to the user. Stores in use by the customer are highlighted in pale green, thus not in use are highlighted in grey.

```
private void populateListViewStores() {
    Log.d(CLASS_NAME, "\t: populateListViewStores for all stores ...");
    int i = 0;
    for (Store store : stores) {
        storeImages[i] = Integer.parseInt("R.drawable"+store.getLogo());
        storeNames[i] =  store.getName().toString();
        i++;
    }
    storeItems = new ArrayList<StoreItem>();
    for (int x=0; x<storeImages.length; x++){
        StoreItem item = new StoreItem(storeImages[x], storeNames[x]);
        storeItems.add(item);
    }
    storeListView = (ListView) findViewById(R.id.storeList);
    storeListView.setAdapter(new StoreAdapter(this, R.layout.list_item_image_store,
       storeItems));
    storeListView.setOnItemClickListener(this);
}
```

If a Store which is not in use is selected then the option is given to the user to add that Store to their active listing in onItemClick :

```
public void onItemClick(AdapterView<?> parent, View view, int position, long _id) {
```

```
    Log.d(CLASS_NAME, "\t: onItemClick for selected row ...");
    int imageId = storeImages[position];
    String store = storeNames[position];
    StoreForCustomer storeForCustomer = new StoreForCustomer();
    int storeId = databaseAdapter.getStoreIdUsingStoreName(store);
    int points = databaseAdapter.getStorePointsBalanceUsingStoreId(storeId);
    Intent intentStoreListing = new Intent(this, GenerateZXingCodeActivity.class);
    intentStoreListing.putExtra(STORE_IMAGE, imageId);
    intentStoreListing.putExtra(STORE_NAME, store);
    intentStoreListing.putExtra(STORE_POINTS, points);
    startActivity(intentStoreListing);
}
```

2.7.4 LAYOUT : ACTIVITY_DISPLAY_STORE_LISTING.XML

This uses a <ListView> which is populated using the image and name from list_item.xml :

```
<ListView
     android:id="@+id/storeList"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:gravity="center_vertical"
     android:padding="10dp" > </ListView>
```

Other files used are explained in the section : Store ListView Files.

2.8 ACTIVITY – STORE STATUS [STORE_NAME] (NEW & EXISTING USER)

2.8.1 WHAT THIS ACTIVITY DOES?

This displays current points available and also any money-off or %-off vouchers which may be redeemed for the selected store. If any available vouchers is selected by the customer then a QR-code or Bar-code (depending) on user selected is generated in the next activate.

2.8.2 VIEW : MOCK-UP -V- ACTUAL

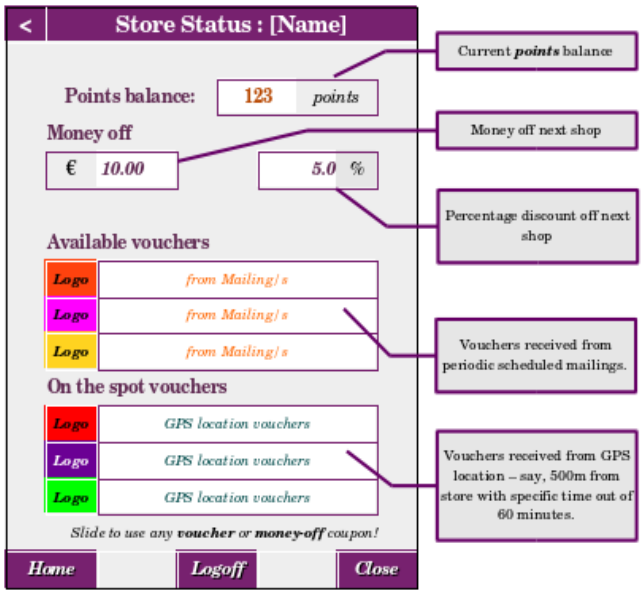
Mock-up	Actual
	

Table 2:9 – Store Status → available points and offers.

Difference/s	Common updates (see : App Flow – Common Discrepancies). 1. GPS vouchers options removed as not enough time to implement this functionality. It would be a good add-on in the future.
Reason/s	1. Insufficient time.

2.8.3 CODE : DISPLAYSTORESPECIFICINFO.JAVA

2.8.4 LAYOUT : ACTIVITY_DISPLAY_STORE_SPECIFIC_INFO.XML

Again, I used a <TableView> inside a <ScrollView>, this will facilitate screen scrolling, if needed.

In the first row, we have the Store logo → <ImageView> and Store name → <TextView>. In the next row, we have Curent Balance and points available (both → <TextView>). In the third row, we have Available Stores → <TextView> in column 1 and a listing of available stores in column 2. If a Store is selected, then the activity to open the QR-code and Bar-code generation activity.

[Cancel]	was formatted using *drawable/cell_round_green.xml,* and had an onClick listener assigned to return to the calling activity.

As we are now in the Stores section, buttons and title backgrounds are now formatted in **green.**

2.9 ACTIVITY – BAR-CODE (NEW & EXISTING USER)

2.9.1 WHAT THIS ACTIVITY DOES?

This displays the QR-code or the Bar-code for the selected voucher.

2.9.2 VIEW : MOCK-UP -V- ACTUAL

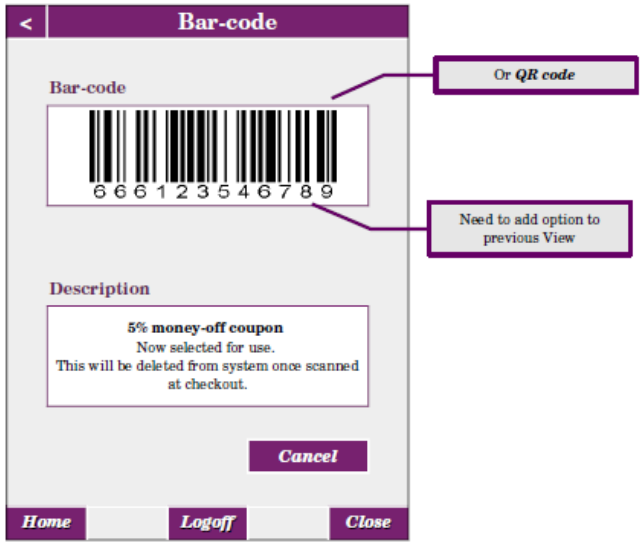
Mock-up	Actual
	

Table 2:10 – Generate QR-Code & Bar-Code for selected item.

Difference/s	Common updates (see : App Flow – Common Discrepancies). 1. Addition of both QR-code and Bar-code to same screen. 2. Title bar is disabled. 3. A **[Back]** button was also added.
Reason/s	1. Either may be used depending on store facilities. 2. Not needed in this activity. 3. The user may now cancel the transaction and return to the previous screen.

2.9.3 CODE : GENERATEZXINGCODEACTIVITY.JAVA

Title bar / Activity bar are disabled for this activity using :

```
requestWindowFeature(Window.FEATURE_NO_TITLE);
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

```
setContentView(R.layout.activity_generate_zxing_code);
```

Depending on the option chosen in the previous activity the QR-code and the Bar-code for the voucher are displayed for use. The following is the code for Bar-code (differences are marked in ***yellow***).

```
private void generateQRCode(String data) {
  Log.d(CLASS_NAME, ":\tgenerateQRCode ...");
  int xRange = 150, yRange = 150;
  int colorBack = 0xFF000000, colorWhite = 0xFFFFFFFF;
  Writer qrWriter = new QRCodeWriter();
  String finalData = Uri.encode(data, "utf-8");
  try {
     BitMatrix bitMatrix = qrWriter.encode(finalData, BarcodeFormat.QR_CODE, xRange, yRange);
     Bitmap imageBitmap = Bitmap.createBitmap(xRange, yRange, Bitmap.Config.RGB_565);
     for (int x = 0; x < xRange; x++) {
        for (int y = 0; y < yRange; y++) {
           imageBitmap.setPixel(x, y, bitMatrix.get(x, y) ? colorBack : colorWhite);
        }
     }
     ImageView imageview = (ImageView) findViewById(R.id.ivQRCode);
     if (imageBitmap != null) imageview.setImageBitmap(imageBitmap);
  } catch (WriterException e) {
     e.printStackTrace();
  }
}
```

For Bar-Code generation we have :

- xRange → '***180'*** pixels
- yRange → '***40'*** pixels,
- writer → '*MultiFormatWriter barWriter = new MultiFormatWriter();*'"
- BarcodeFormat → '*BarcodeFormat.CODE_128'*

Obviously, the log method reference and the id of the imageView are also amended.

2.9.4 LAYOUT : ACTIVITY_GENERATE_ZXING_CODE.XML

<ScrollView> containing the logo and name of the store together with the <ImageView> of QR-code and then <Image View> of Bar-code. Finally a **[Back]** button which returns to ***DisplayStoreSpecificInfoActivity***.

2.10 COMMON FILES USED

2.10.1 XML FILES

values/ strings.xml

This is a listing of all strings used in the project : Button, CheckBox, EditText, TextView, Hint.

Some strings were formatted for HTML display … , other strings used formatting via the layout .xml file (TextSize, TextAppearance, etc). This meant that theoretically in a real app – any updates to the contents of or display of any string could be completed by a UX engineer.

```
<!-- HINTS - for equivalent <EditText> fields
repeat login, register login, register customer details,  -->
<string name="hintUserNameToLogin">Confirm/Create : login name</string>
<string name="hintExistingUserLoginPassword">Insert : existing password</string>
<string name="hintRegisterUserPasswordToLogin">Create : new password</string>
<string name="hintConfirmUserPasswordToLogin">Confirm : new password</string>
<string name="hintCustomerNameFamily">Enter : Surname / Family name</string>
<string name="hintCustomerNameFirst">Enter : First name / Use name</string>
<string name="hintAddress1">Address : first line + house number</string>
<string name="hintAddress2">Address : second line e.g.: area</string>
<string name="hintAddress3">Address : third line e.g.: town</string>
<string name="hintAddress4">Address : county</string>
<string name="hintMobile">Enter : Irish mobile number</string>
<string name="hintEmail">Enter : email address</string>

<!-- BUTTONS -
home, repeat login, register login, register customer details, register T&Cs -->
<string name="btnCancel">Cancel</string>
<string name="btnClose">Close</string>
<string name="btnRegister">Register</string>
<string name="btnAgreeYes">I Do \nAgree</string>
<string name="btnAgreeNo">I Don\'t \nAgree</string>
<string name="btnSignIn">Sign In</string>
<string name="btnSubmit">Submit</string>
<string name="btnClearAll">Clear All</string>
```

values/ colors.xml

This file contains all colors used in the project. Having a common file ensured that any variants in color applied to all usages though one update.

```
<?xml version="1.0" encoding="utf-8"?>
<!--...-->

<resources>
    <color name="black">#000000</color>
    <color name="blue">#0000FF</color>
    <color name="blue_dark">#000099</color>
    <color name="cyan">#00FFFF</color>
    <color name="grey">#bdbdbd</color>
    <color name="grey_pale">#f2f2f2</color>
    <color name="grey_dark">#404040</color>
    <color name="green">#008000</color>
    <color name="green_bright">#00FF00</color>
    <color name="green_dark">#004000</color>
    <color name="orange">#FFA500</color>
    <color name="purple">#800080</color>
    <color name="red_dark">#660000</color>
    <color name="teal">#008080</color>
    <color name="white">#FFFFFF</color>
    <color name="yellow">#FFFF00</color>
</resources>
```

Table 2:11 – Common XML files in the app.

Rounded buttons were used throughout to ensure continuity throughout the app and to provide a common *look-n-feel*. The easiest way to do so was to define a drawable shape, with specific color depending on where is app flow customer is.

drawable/ cell_round_green.xml

Green was used for stores

```
<?xml version="1.0" encoding="UTF-8"?>

<!--...-->

<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="@color/green" />
    <corners android:radius="20px" />
    <padding android:bottom="10dp" android:left="10dp" android:right="10dp" android:top="10dp" />
</shape>
```

drawable/ cell_round_purple.xml

Purple was used for existing customers.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="@color/purple" />
```

drawable/ cell_round_navy.xml

Navy was used for new customers.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="@color/blue_dark" />
```

drawable/ cell_round_black.xml

Black was used for Lunar Stores references.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="@color/black" />
```

drawable/ cell_round_grey.xml

Grey was used for Cancel, No, etc.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="@color/grey" />
```

drawable/ cell_round_navy_bullet.xml	This was used to fake the bullets for the Terms & Conditions activity, only difference between this and *drawable/cell_round_navy.xml* is the size of the padding "10dp" → "2dp".
	`<shape xmlns:android="http://schemas.android.com/apk/res/android">`
	`  <solid android:color="@color/blue_dark" />`
	`  <corners android:radius="10px" />`
	`  <padding android:bottom="2dp" android:left="5dp" android:right="5dp" android:top="2dp" />`
	`</shape>`

Table 2:12 – Common table titles and button formatting XML files in the app.

2.10.2 SQLite Files

DataBaseHelper.java	This works on the database itself for onCreate and onUpgrade (structure changes, version incremented, version decremented).
	`public void onCreate(SQLiteDatabase db) {`
	`  Log.d(CLASS_NAME, ":\t onCreate - create three tables in SQLite database...");`
	`  db.execSQL(CREATE_TABLE_STORE);`
	`  db.execSQL(CREATE_TABLE_CUSTOMER);`
	`  db.execSQL(CREATE_TABLE_SHOP_FOR_CUSTOMER);`
	`}`
	and
	`public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {`
	`  Log.d(CLASS_NAME, ":\t onUpgrade inc. recreate three tables for SQLite ...");`
	`  db.execSQL("DROP TABLE IF EXISTS " + TABLE_STORE);`
	`  db.execSQL("DROP TABLE IF EXISTS " + TABLE_CUSTOMER);`
	`  db.execSQL("DROP TABLE IF EXISTS " + TABLE_SHOP_FOR_CUSTOMER);`
	`  onCreate(db);`
	`}`
	Basically, the SQLite Schema!

DatabaseAdapter.java	This is responsible for managing the SQLite database data. It is used by the app to insert, update, delete and query data from the SQLite database tables : customer.tb, store.tb and store4customer.tb. Basically, CRUD functionality as well as open and close the database once done.
	For example : delete data from STORE.TB using the PK (id) for the table :
	`public void deleteStore(long store_id) {`
	`  Log.d(CLASS_NAME, ":\t delete store for specified store id ...");`
	`  SQLiteDatabase db = dbHelper.getWritableDatabase();`
	`  db.delete(TABLE_STORE, KEY_ID + " = ?", new String[] { String.valueOf(store_id) });`
	`}`

Table 2:13 – 3-Level SQLite files as applied.

2.10.3 STORE LISTVIEW FILES

StoreAdapter.java

Adapter for the store items for the listing :

```
public View getView(int position, View view, ViewGroup parent) {
  StoreHolder storeHolder = null;

  LayoutInflater layoutInflater = (LayoutInflater)
     context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
  if (view == null) {
     view = layoutInflater.inflate(R.layout.list_item, null);
     storeHolder = new StoreHolder();
     storeHolder.image_icon = (ImageView)
        view.findViewById(R.id.image_icon );
     storeHolder.text_store = (TextView)
        view.findViewById(R.id.text_store );
     view.setTag(storeHolder);
  } else {
     storeHolder = (StoreHolder) view.getTag();
  }
  StoreItem storeItem = getItem(position);
  storeHolder.image_icon.setImageResource(storeItem.icon);
  storeHolder.text_store.setText(storeItem.store );

  return view;
}
```

StoreItem.java

Constructor for each store as listed in the activity DisplayStoreListingActivity.

list_item_image_store.xml

This contains a placeholder for each row of the store listing for image on left-hand side and store name on right-hand side of each row:

```
<ImageView
    android:id="@+id/image_icon"
    android:layout_width="80dp"
    android:layout_height="80dp"
    android:layout_alignParentBottom="true"
    android:layout_alignParentTop="true"
    android:layout_marginBottom="5dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="5dp"
    android:contentDescription="@string/icon" > </ImageView>

<TextView
    android:id="@+id/text_store"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_alignParentBottom="true"
    android:layout_alignParentTop="true"
    android:gravity="center_vertical|center_horizontal"
    android:padding="5dp"
    android:textSize="30sp"
    android:textStyle="bold"></TextView>
```

list_item_voucher.xml

This again uses a <ListView>. In this instance, only a <TextView> is present :

```
<TextView
    android:id="@+id/textviewVoucher"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_alignParentBottom="true"
    android:layout_alignParentTop="true"
    android:gravity="center_vertical|center_horizontal"
    android:padding="5dp"
    android:textColor="@color/green"
    android:textSize="30sp"
    android:textStyle="bold|italic"></TextView>
```

Table 2:14 – ListView files as used by the Store activities.

2.10.4 GLOBALVARIABLECUSTOMERID.JAVA

Define variable int ***custId*** in applicable context and may then use it as global variable. Here customer id is used to reference CUSTOMER.TB and STORE4CUSTOMER.TB for that specific customer → user currently logged.

```
package org.dwan.paula.lunarenterprises;

import android.app.Application;

public class GlobalVariableCustomerId extends Application {
    private  int custId;
    public int getCustId() { return custId; }
    public void setCustId(int custId) { this.custId = custId; }
}
```

This is referenced in AndroidManifest.xml <Application > section :

```
android:name="org.dwan.paula.lunarenterprises.GlobalVariableCustomerId"
```

The global variable ***custId*** may then be called and set in SignInActivity.java :

```
Customer customer = new Customer();
final GlobalVariableCustomerId globalVariableCustomerId = (GlobalVariableCustomerId)
     getApplicationContext();

customer =  databaseAdapter.getCustomerUsingUserName(etUser.getText().toString());
globalVariableCustomerId.setCustId(customer.getId());
```

read (get) in any other activity and thus used :

```
final GlobalVariableCustomerId globalVariableCustomerId = (GlobalVariableCustomerId)
     getApplicationContext();

String currentCustomer = globalVariableCustomerId.getCustId();
```

3 TECHNOLOGIES

3.1 TECHNOLOGIES PLANNED

3.1.1 SQLITE DATABASE SOFTWARE LIBRARY OVERVIEW

SQLite is a software library that implements a self-contained, server-less, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain. [1]

Each Android App has its own SQLite database and is available only to that app and no others on the device thus enabling a more secure implementation.

SQLite as other SQL based database systems also adheres to the SQL principles of :

Atomicity	All or nothing – if one part of the transaction fails then all fails (e.g.: if customer surname is invalid or not present then no data for that new customer is written to `CUSTOMER.TB`.
Consistency	For each transaction, the database is in a valid state for before and after the transaction is implemented.
Isolation	Running transactions simultaneously is the same as running one after the other – each is implementation independently of the other.
Durability	Once a transaction is completed, it is present in the database regardless of device system failures, etc.

3.1.2 ZXING OVERVIEW

ZXing ("zebra crossing") is an open-source, multi-format 1D/2D barcode image processing library which is implemented using Java. There is functionality for other languages also. It was initially developed in Japan about 1994 for car manufacturing.

Formats supported include 1-D product, 1-D industrial and 2-D. For this project, I used :

- QR-Code → 2-D = QR-Code
- Bar-Code → 1-D industrial = Code-128

Android Components present in the latest release and those needed for **this** Android project are :

core	The core image decoding library, and test code
javase	JavaSE-specific client code
android	Android client Barcode Scanner Barcode Scanner
androidtest	Android test app, ZXing Test
android-integration	Supports integration with Barcode Scanner via Intent
android-core	Android-related code shared among android, androidtest, glass
glass	Simple Google Glass application
zxingorg	The source behind zxing.org
zxing.appspot.com	The source behind web-based barcode generator at zxing.appspot.com

There are certain permissions which are required when ZXIng is incorporated into an app for bar-code and/or QR-code generation or usage. These are implemented in the AndroidManifest.xml file and are :

android.permission. CAMERA	READ → Use the device's photo camera for reading the barcode **e.g.:** *<uses-permission android:name="android.permission.CAMERA"/>*
android.permission. VIBRATE	READ / WRITE → Report that the product was successfully read / written
android.permission. FLASHLIGHT	READ → Better view of the barcode
android.permission. WAKE_LOCK	READ / WRITE → Maintain the application open, until the scanning operation is successfully done
android.permission. WRITE_EXTERNAL_STORAGE	READ / WRITE → Allow app to write bitmap to device for QR-code or for Bar-code.

Table 3:1 – android.permissions options available.

I implemented ZXing to generate the bar-code and/or the QR-code for the voucher or money-off coupon.

ZXing may also be used to read either bar-code or QR-code using the phone's or the tablet's camera. This is not implemented for this app but could be in the future, if commercially developed, to say read once vouchers placed at strategic positions in the store.

3.1.3 Encryption of User Data

Boton	✗	http://botan.randombit.net/ This was ruled out as this is aimed at C++ developers. Even though, it is released under a very open free BSD-2 license (see : http://botan.randombit.net/license.html for more information), it is not applicable to Android development.
SEE	✗	SQLite Encryption Extension http://www.hwaci.com/sw/sqlite/see.html Again, this was ruled out due to licence fees – a perpetual licence costs US$ 2,000.
SQL ciper	✓	https://www.zetetic.net/sqlcipher/ https://www.zetetic.net/sqlcipher/sqlcipher-for-android/ → how to use in Android app This was the one chosen as it is basically an extension to SQLite. SQLCipher also employs *256-bit AES encryption* of database files and it works on Linux and is free of charge (open source licence)..
SQLiteCrypt	✗	http://sqlite-crypt.com/documentation.htm This was ruled out as this is licensed. The licence fee is currently US$128 ("one platform binary").
WxSQLite	✗	http://sourceforge.net/projects/wxsqlite/ This is suited to MAC and to WOS operating systems as I used Ubuntu, it is not applicable.

Table 3:2 – Encryption options evaluated

3.1.4 ENSURING SECURE LOGIN – USERS

Linux operating system	Android devices use a Linux based operating system, this means that groups, users and also specific verification to execute files based on user and group.
Application signing	Also, the developer releases the app with a specific certificate. Once published, each Android application is given its own unique user ID, generated at installation. Anytime an app tries to do something it doesn't have permission to do, it results in a security exception and it halts. When published on the market place, the app details what permissions are needed → access to contact lists, browser history, etc.
Mandatory application sandbox	Isolate code execution from other apps on the device. This also separates app data for each app.
Secure interprocess communication	Various technologies including Linux which are used to minimize risks which may arise due to memory management errors.
Application-defined permissions	Apps restrict access to app specific data.
user granted permissions	Users wish to restrict access and user data unless absolutely necessary. Users also wish to precent unwanted access to system features.

Table 3:3 – Core Security features of Android development.

Secure Sockets Layer (SSL) is probably the easiest and most recognizable secure login facility available today.Typically, a server is configured with a certificate containing a public key : private key pair. The client and server communicate once the server proves that it has the private key (PKI).

3.2 TECHNOLOGIES ACTUALLY USED

3.2.1 SQLITE DATABASE

SQLite is based on SQL which is explained as follows :

ⓘ Structured Query Language (SQL) is a language used to create new, access existing, query existing, and update existing relational databases. A relational database has one or more tables with a unique identifier for each row of all tables.

Data is retrieved from a table or multiple tables using the main SQL commands [5] :

- SELECT .....extracts data from a database
- UPDATE .....updates data in a database
- DELETE .....deletes data from a database
- INSERT INTO .....inserts new data into a database
- CREATE DATABASE .....creates a new database
- ALTER DATABASE .....modifies a database
- CREATE TABLE .....creates a new table
- ALTER TABLE .....modifies a table
- DROP TABLE .....deletes a table

SQLite was used as it is a standalone database system which is compact and suited to use on Android or iOS devices. There are two .java files used to define and use the SQLite database and these are explained in the section : SQLite Files.

3.2.1.1 SQL TABLE AS IMPLEMENTED – CUSTOMER.TB

CUSTOMER.TB contains customer information for every customer : name, address, sign-up date, contact information and confirmation that the contact information may or may not be used for additional offers or marketing. This table also stores the customer's login name and login password and also interacts with both tables STORE.TB and with store4customer.TB. Each customer has a unique identifier in the table – the ***primary key*** = CUSTOMER_ID. This is an automatically incremented integer. Finally, CUSTOMER.TB contains boolean confirmation if the customer has signed up to shop1, shop2, etc. for vouchers and money-off.

CUSTOMER.TB was created using SQLite within the app implementation in ***LunarDatabaseHelper.java***:

```
private static final String CREATE_TABLE_CUSTOMER =
"CREATE TABLE " + TABLE_CUSTOMER + "(" + KEY_ID + " INTEGER PRIMARY KEY, "
+ KEY_NAME_1 + " TEXT NOT NULL, " + KEY_NAME_2 + " TEXT NOT NULL, "
+ KEY_ADDRESS_1 + " TEXT NOT NULL, "+ KEY_ADDRESS_2 + " TEXT NOT NULL, "
+ KEY_ADDRESS_3 + " TEXT, " + KEY_ADDRESS_4 + " TEXT NOT NULL, " + KEY_EMAIL + " TEXT, "
+ KEY_MOBILE + " TEXT, " + KEY_EMAIL_OK + " TEXT, " + KEY_MOBILE_OK + " TEXT, "
+ KEY_THIRD_PARTY_OK + " TEXT, " + KEY_LOGIN + " TEXT NOT NULL, "
+ KEY_PASSWORD + " TEXT NOT NULL, " + KEY_CREATED_AT + " DATETIME" + ")";
```

using the call :

```
db.execSQL(CREATE_TABLE_CUSTOMER);
```

Structure is as follows and the table was pre-populated with five customers :

Database : lunarstores.db								
Table : customer.tb								
Column ID	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Name	**_id**	**name1**	**name2**	**address1**	**address2**	**address3**	**address4**	**email**
Type	*Integer*	*TEXT NOT NULL*	*TEXT NOT NULL*	*TEXT NOT NULL*	*TEXT NOT NULL*	*TEXT*	*TEXT NOT NULL*	*TEXT*
Default Value	*null*	*null*	*null*	*null*	*null*	*null*	*null*	*null*
PK	*T*	*F*	*F*	*F*	*F*	*F*	*F*	*F*
FK	*F*	*F*	*F*	*F*	*F*	*F*	*F*	*F*
Value 1	0	John	Ryan	12 Main St.,	Douglas	*null*	Cork	jryan@gmail.com
Value 2	1	Amy	Collins	123 Rock Ave,	Cork	*null*	Cork	acollins@gmail.com
Value 3	2	Emily	Blackwell	555 Roswell Ave.	Blessington	*null*	Wicklow	emily@gmail.com
Value 4	3	Max	Roberts	25 Second St.,	Naas Rd.,	Blessington	Wicklow	mroberts@gmail.com
Value 5	4	Cathy	Jones	22 O'Connell St	Dublin 1	*null*	Dublin 1	jones123@gmail.com

Column ID	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Name	**mobile**	**emailOk**	**mobileOk**	**thirdPartyOk**	**nameLogin**	**passwordLogin**	**createdAt**
Type	*TEXT*	*TEXT*	*TEXT*	*TEXT*	*TEXT NOT NULL*	*TEXT NOT NULL*	*TEXT NOT NULL*
Default Value	*null*	*null*	*null*	*null*	*null*	*null*	*null*
PK	*F*	*F*	*F*	*F*	*F*	*F*	*F*
FK	*F*	*F*	*F*	*F*	*F*	*F*	*F*
Value 1	087-7766321	1	1	1	jryan	password	*getCurrentDate*
Value 2	085-7654321	1	0	0	acollins	password	*getCurrentDate*
Value 3	085-7654000	0	1	0	eblackwell	password	*getCurrentDate*
Value 4	086-9988771	0	0	1	robertsm	password	*getCurrentDate*
Value 5	null	0	0	0	jonesc	password	*getCurrentDate*

3.2.1.2 SQL TABLE AS IMPLEMENTED – STORE.TB

STORE.TB contains information on each shop which has signed up the scheme : name, logo, address, contact number, contact email, name of main contact, name of backup contact, date shop joined the scheme and date shop left scheme. Each shop has a unique identifier in the table – the ***primary key*** = STORE_ID. This is an automatically increment integer. Once a store leaves the scheme and thus database then the points collected alread no longer apply.

The table STORE.TB was created using SQLite during app implementation in ***LunarDatabaseHelper.java*** :

```
private static final String CREATE_TABLE_STORE =
"CREATE TABLE " + TABLE_STORE + "(" + KEY_ID + " INTEGER PRIMARY KEY, "
+ KEY_NAME + " TEXT NOT NULL, " + KEY_LOGO + " TEXT NOT NULL, "
+ KEY_CONTACT_NAME + " TEXT NOT NULL, " + KEY_CONTACT_EMAIL + " TEXT NOT NULL, "
+ KEY_CONTACT_PHONE + " TEXT NOT NULL, " + KEY_CREATED_AT + " DATETIME" + ")";]
```

using the call :

```
db.execSQL(CREATE_TABLE_STORE);
```

This was pre-populated using five stores :

Database : lunarstores.db							
Table : store.tb							
Column ID	0	1	2	3	4	5	6
Name	_id	name	logo	contactName	contactEmail	contactNumber	createdAt
Type	Integer	TEXT NOT NULL	TEXT NOT NULL	TEXT NOT NULL	TEXT NOT NULL	TEXT NOT NULL	TEXT NOT NULL
Default Value	null	null	null	null	null	null	null
PK	T	F	F	F	F	F	F
FK	F	F	F	F	F	F	F
Value 1	0	Alpha Stores	logo_alpha	Joe Walshe	alphastores@gmail.com	085-1122345	*getCurrentDate*
Value 2	1	Beta Stores	logo_beta	Peter Roberts	betastores@gmail.com	087-2323543	*getCurrentDate*
Value 3	2	Orange Stores	logo_orange	Emma Williams	info@orange-stores.com	085-5551234	*getCurrentDate*
Value 4	3	$$$-Stores	logo_dollar	Trixie McRoy	info@dollar-stores.com	086-1234555	*getCurrentDate*
Value 5	4	€€€-Stores	logo_euro	Josh Williamson	info@euro-stores.com	085-1221343	*getCurrentDate*

3.2.1.3 SQL TABLE AS IMPLEMENTED – STORE4CUSTOMER.TB

`STORE4CUSTOMER.TB` contains a direct 1-1 link between a shop in `STORE.TB` and a customer in `CUSTOMER.TB` using the ***primary keys*** of `CUSTOMER_ID` and `STORE_ID` respectively. `STORE4CUSTOMER.TB` has a ***primary key*** = `SHOP4_ID` and again, this is an automatically increment integer.

The table `STORE4CUSTOMER.TB` was created using SQLite during app implementation in ***LunarDatabaseHelper.java*** as follows :

```
private static final String CREATE_TABLE_SHOP_FOR_CUSTOMER =
"CREATE TABLE " + TABLE_SHOP_FOR_CUSTOMER + "(" + KEY_ID + " INTEGER PRIMARY KEY, "
+ KEY_STORE_ID + " INTEGER, " + KEY_CUSTOMER_ID + " INTEGER, "; + KEY_POINTS_BALANCE
+ " INTEGER, " + ")";
```

using the call :

```
db.execSQL(CREATE_TABLE_SHOP_FOR_CUSTOMER);
```

Again, the table was pre-populated with two shops allocated to each customer :

Database : lunarstores.db					
Table : store4customer.tb					
Column ID	0	1	2	3	4
Name	_id	customerId	StoreId	StorePoints	createdAt
Type	Integer	Integer	Integer	Integer	TEXT NOT NULL
Default Value	null	null	null	null	null
PK	T	F	F	F	F
FK	F	customer.tb : _id	store.tb : _id	F	F
Value 1	0	1	1	100	*getCurrentDate*
Value 2	1	2	1	200	*getCurrentDate*
Value 3	2	3	2	300	*getCurrentDate*
Value 4	3	4	2	400	*getCurrentDate*
Value 5	4	5	4	500	*getCurrentDate*
Value 6	5	1	2	600	*getCurrentDate*
Value 7	6	2	3	700	*getCurrentDate*
Value 8	7	3	4	800	*getCurrentDate*
Value 9	8	4	5	900	*getCurrentDate*
Value 10	9	5	5	1000	*getCurrentDate*

Customer	Store
jryan	*Alpha Stores*
acollins	*Alpha Stores*
eblackwell	*Beta Stores*
robertsm	*Beta Stores*
jonesc	*Orange Stores*
jryan	*Orange Stores*
acollins	*$$$-Stores*
eblackwell	*$$$-Stores*
robertsm	*€€€-Stores*
jonesc	*€€€-Stores*

When a customer uses the app, all rows in the table `STORE4CUSTOMER.TB` containing that `CUSTOMER_ID` and a valid `STORE_ID` indicate that store is available for use by that customer. All other stores are available for current or future selection. If another store is activated by the customer then a new row is added to the table and the customer may start collecting points having an initial value of 0 (nil) points. This balance is increased if the customer shops in that store or decreased if the customer uses available points to purchase a % or money-off voucher.

3.2.2 ZXING QR-CODE & BAR-CODE GENERATION

Also, ZXing works on Android via Intents or by referencing the ZXing library. Intents are easier to implement but the customer must have ZXing preinstalled on his/her device. This involved the customer agreeing to a 3rd party product and additional download before he/she even gets to use ***Lunar Stores.*** Alternatively directly reference the libraries, and this was the method implemented for this project. In order to use ZXing libreary, add **this** line to the dependencies section of the gradle.build file (ZXing is present in the Maven Public Repositories and may be retrieved from there) :

```
dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  compile 'com.android.support:appcompat-v7:21.0.3'
  compile 'com.google.zxing:core:3.1.0'
}
```

and then import the library in the usual fashion in the activity :

```
  import com.google.zxing
```

There are minor differences between QR code and Bar-code Generation, mainly in image size, writer/reader, depending on where the *-code is a voucher generated for use in a store or is a code being read to increate customer points. Both, however are black and white so minimal RGB is required.

3.2.2.1 ZXING QR-CODE GENERATION

QR-Code uses *com.google.zxing.Writer* to create a new *QRCodeWriter(),* The *BarCodeFormat* is *QR_CODE* and of course the resulting Bitmap is square *150x150* pixels.

Creation of the Bitmap is the same as that for the Bar-Code pixels are generated for x co-ordinate, and y co-ordinate for the full square and each pixel is black or white. The resulting bitmap is then displayed to the user for use.

ARGB_8888 is recommended for the Config of the resulting bitmap but as this is a Black & White image, pixelation will not be lost if *RGB_565* is used instead.

3.2.2.2 ZXING BAR-CODE GENERATION

Bar-Code uses *com.google.zxing.MultiFormatWriter* to create a new MultiFormat*Writer(),* The *BarCodeFormat* is *CODE_128* and of course the resulting Bitmap is rectangular *180x40* pixels.

3.2.3 ENCRYPTION OF USER DATA

Due to the reasons detailed in section , I choose to use SQLCipher to encrypt user data in the SQLite database.

Advantages	1. Strong encryption → 256-bit Advanced Encryption Standard (AES), Rijndael algorithm 2. Open-source so thus has advantage of many developers working on improvements, as well as company developers 3. Works in tandem with as standard Adnroid database funcitons 4. Mature technology
Disadvantages	1. APK will be much larger in size 2. Database access (RW) will be slower as all data is encrypted, thus the app will be slower. Will this impact user satisfaction or is encryption perceived as worth it? Expect 10%-30% reduction depending on app size and number of database read|write. If speed is a factor, then there is always AES 128-bit and AES 192-bit. 3. The encryption key will need to be retained safely. Also, is the same key used for all instances of the app or is one created per device? 4. The USA or some other countries are less inclined to support strong encryption than is Ireland.

Using SQLCipher into an Android Studio project is relatively painless. Instructions follow :

Integration into Android Studio

1. Download SQLCipher *(ZIP format)* from zetetic.net or clone from from git :

```
https://www.zetetic.net/sqlcipher/open-source/
git clone https://github.com/sqlcipher/android-database-sqlcipher.git
```

2. In app/src/main directory, create two sub-directories as follows :

```
app/src/main/jniLibs
app/src/main/assets
```

3. Extract contents of */assets* directory (including all sub-directories and contents) from ***sqlcipher-for-android-community-v3.3.0.zip*** to *app/src/main/assets.*
4. *Extract contents of /libs directory (including all sub-directories and contents) from* ***sqlcipher-for-android-community-v3.3.0.zip*** *to app/src/main/jniLibs.*

   Also add : ***guava-r09.jar*** and ***commons-codec.jar*** to *app/src/main/jniLibs* – these do not seem to form part of the .zip file and will need to be searched for online.
5. Amend gradle.build to refer to SQLCipher, by adding the following line :

```
compile fileTree (dir: 'libs', include: ['*.jar'])
```

Replace SQLite with SQLCipher

1. Replace generic SQLite imports with SQLCipher imports :

```
//   WAS
     import android.database.Curson;
     import android.database.SQLException;
     import android.database.sqlite.SQLiteDatabase;
//   NOW
     import net.sqlciper.Curson;
     import net.sqlciper.SQLException;
     import net.sqlciper.database.SQLiteDatabase;
```

2. Amend all calls to open/create database :

```
//   WAS
     getWritableDatabase();
//   NOW
     getWritableDatabase('my_sequre_key');
```

3. Initialize SQLCipher (first command to run) :

```
import net.sqlciper.database.SQLiteDatabase

SQLiteDatabase.loadLibs(context);
```

3.2.4 Ensuring Secure Login – Users (Basic)

Unique user name is required for each user. If a user created a second account with the same contact details, he/she was permitted to do so. For future iterations, it would be best to confirm that this was completed by email or SMS.

Also minimum requirements for user login name length and also password are required. I don't think that applying more stringent sign-in requirements, for example : alpha-numeric password with no proper words allowed or requiring users to sign-in in provate would be viable. This is a consumer app, one that is designed to be used in a public venue – hopefully frequently. It will not be practical to apply as strong safe-guards as Banks.

3.2.5 ENSURING SECURE LOGIN – USERS (ANDROID DEVELOPMENT)

Some of the areas raised in section *Ensuring Secure Login – Users* for consideration are detailed as follows

Mandatory application sandbox	This is a standalone app and the executable code and the local database are independent of any other apps on the device. No other app should have acess to the data Also once the user logs in the login name | password pairing is no longer used, the customer id is used instead. This is unique and non-transferable to another person. It also is numeric and has no relation to the customer.
Application-defined permissions	Content providers may be restricted to the app or share information with other apps. Restrict access to the app using one line in the AndroidManifes.xml file : `android:exported="false"` Be careful if your app permits use of external storage → only store non-sensitive information (e.g.: general information but not database information or login details)
user granted permissions	This may be set using **(i)** System properties from *ACCESS_CHECKING_PROPERTIES* to *WRITE_VOICEMAIL* or **(ii)** <permission> option : `<permission android:description="string resource" android:icon="drawable resource" android:label="string resource" android:name="string" android:permissionGroup="string" android:protectionLevel= ["normal"|"dangerous"|"signature"|"signatureOrSystem"] />` Both are set in the AndroidManifest.xml file

3.3 WHAT I WOULD HAVE USED INSTEAD!

There are other database options available but SQLite is one which seems to be preferred for Android delopment due to its low impact on memory during exection and saving. However, I found it somewhat lacking (no Foreign Keys, Triggers had to be manully implemented). This reduced ease of use for non-SQL experienced persons and also increases the overheads on maintenance of a live system.

<TODO> _id applicable?

SQLCipher is designed to work with SQLite and is readily integrated into new or existing code. However, there

4 CHALLENGING / INTERESTING FEATURES

4.1 WHAT I THOUGHT WOULD BE CHALLENGING / INTERESTING?

SQL / SQLite	I expected SQL / SQLite implementation to be of particular interest. Previously, I would have worked on SQL databases in Linux and it is only through this project that I realized how much knowledge I had lost.
ZXing	I expected this to be somewhat difficult to implement as most of the examples and documentation cover reading of QR-Codes and Bar-Codes and not generation.
Encryption	
Security	

4.2 IN THE END – WHAT WAS ?

SQL / SQLite	SQLite implementation was as interesting as I had expected it to be.
ZXing	The actual implementation was reasonably okay, however whether this will work with shop bar-code / QR-code readers is not something I have tested so I don't know if the slight glare on some phone screens would impact use. It may be necessary to dim the screen before hand and reset it after the bar-code / QR-code is read at point of sale.
Encryption	While I looked at SSL
Security	
Other	Simple things → design of logos, store names. Due to licencing I had not be careful not to use any curently registered to an existing company. This proved more difficult than anticipated.

4.3 IF I HAD MY TIME OVER – WOULD I DO THIS PROJECT?

Yes but I would have been more specific on what I would accomplish in the time available. I would also have done more initial research into the technologies I thought appropriate. I also think that this has more commercial suitability that I had orginally envisaged and thus would have been better completed by a team, thus ensuring that more than one viewpoint was taken into consideration. I would also have partnered with a retail outlet for testing purposes, just to make sure it workds as planned. Finally, I would survey shoppers in various age-groups to see exactly what each needs. As what I have designed may not be what the market wants.

5 CONCLUSIONS / FINAL COMMENTS

There are similar app's available on the market. One thing to note is that Android users seem concerned that companies are installing and upgrading apps without users registering for that app. Even if that app is dormant, upgrades are completed. For example; the Sansung app *'Beaming Service for Samsung'*. This goes against privacy claims that many company Android app suppliers advocate and users request.

> *we are not Apple or Google, we will not retain your data without your permission (paraphrased)*

This is the main reason that I design a registration activity as part of this app – if the user does not agree to the T&C's or if he/she de-registers then the app is disabled on his/her Android device. For future implementation, perhaps on de-registration the user can be offered an option to delete the app from the device, similarly this option could also be made available in the T&C activity.

This app if offered commercially would require more back-end support and implementation than I have provided. This is due to Data Protection and security concerns.

It may also be possible to enable a user to apply a bespoke background perhaps sponsoring a local organization or team (10% of points raised by the customer would go to that organization). This could be advertised by the customer through a background or color schema in the teams / schools colors.

3-Level SQLite was applied which would make it relatively easy to separate the database helper and adapter into separate package, this would be advisable for a commercial app. In this implementation, I just created separate packages for customer, store, database, etc. This could be amended with the addition Design Pattern Factories for additon of new users →

- multiple simultaneously – how handled?
- who takes priority if a duplicate login name
- complexity of password required – I would expect a lot of people would use say *'Passw0rd'*, date of birth or the name of a local store for ease of re-call).

6 REFERENCES

Ref.	Information	How used? / What for?
[1]	SQLite database software library https://www.sqlite.org/	Store customer and store information in separate database (.db) files.
[2]	SQLite tutorials http://www.tutorialspoint.com/sqlite/ http://www.vogella.com/tutorials/AndroidSQLite/article.html SQLite – multiple tables http://www.androidhive.info/2013/09/android-sqlite-database-with-multiple-tables/ SQLite – ListView http://www.mysamplecode.com/2012/07/android-listview-cursoradapter-sqlite.html	Introduction to applying SQLite (overview, multiple tables and listview)
[3]	SQLite training https://www.sqlite.org/autoinc.html	Introduction to applying SQLite to project implementation.
[4]	Bar-code / QR-code Generation https://github.com/zxing/ http://codeisland.org/2013/generating-qr-codes-with-zxing/	Code for ZXing Trainign for ZXing *(not really well documented for generation on device)*
[5]	SQL http://www.w3schools.com/sql/default.asp	Introduction to SQL commands and usage.
[6]	Logos	Source material for logo's used for shop logos and thus icons
[7]	ListView – icon & text Assignment 2, referenced *STORE.TB*	ListView of stores for which points are collected or add a store to the list and start collecting points for it.
[8]	Tick-box / Check-box http://examples.javacodegeeks.com/android/core/ui/checkbox-ui/android-checkbox-example/	Tutorial - allow SMS, email & 3rd party contact customer
[9]	SQLCipher https://www.zetetic.net/sqlcipher/ https://www.zetetic.net/sqlcipher/sqlcipher-for-android/	Source material and training for SQLCipher (encryption for SQLite).
[10]	Android Security https://developer.android.com/training/articles/security-tips.html http://developer.android.com/training/articles/security-ssl.html	Built-in security features in Anddroid

SQLCipher licence	**SQLCipher Community Edition distributions are made available under the following BSD style license.** *Copyright (c) 2008-2012 Zetetic LLC* *All rights reserved.* *Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:* • *Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.* • *Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.* • ** Neither the name of the ZETETIC LLC nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.* *THIS SOFTWARE IS PROVIDED BY ZETETIC LLC "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ZETETIC LLC BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.*