Store Visit,GPS & Store Checkin

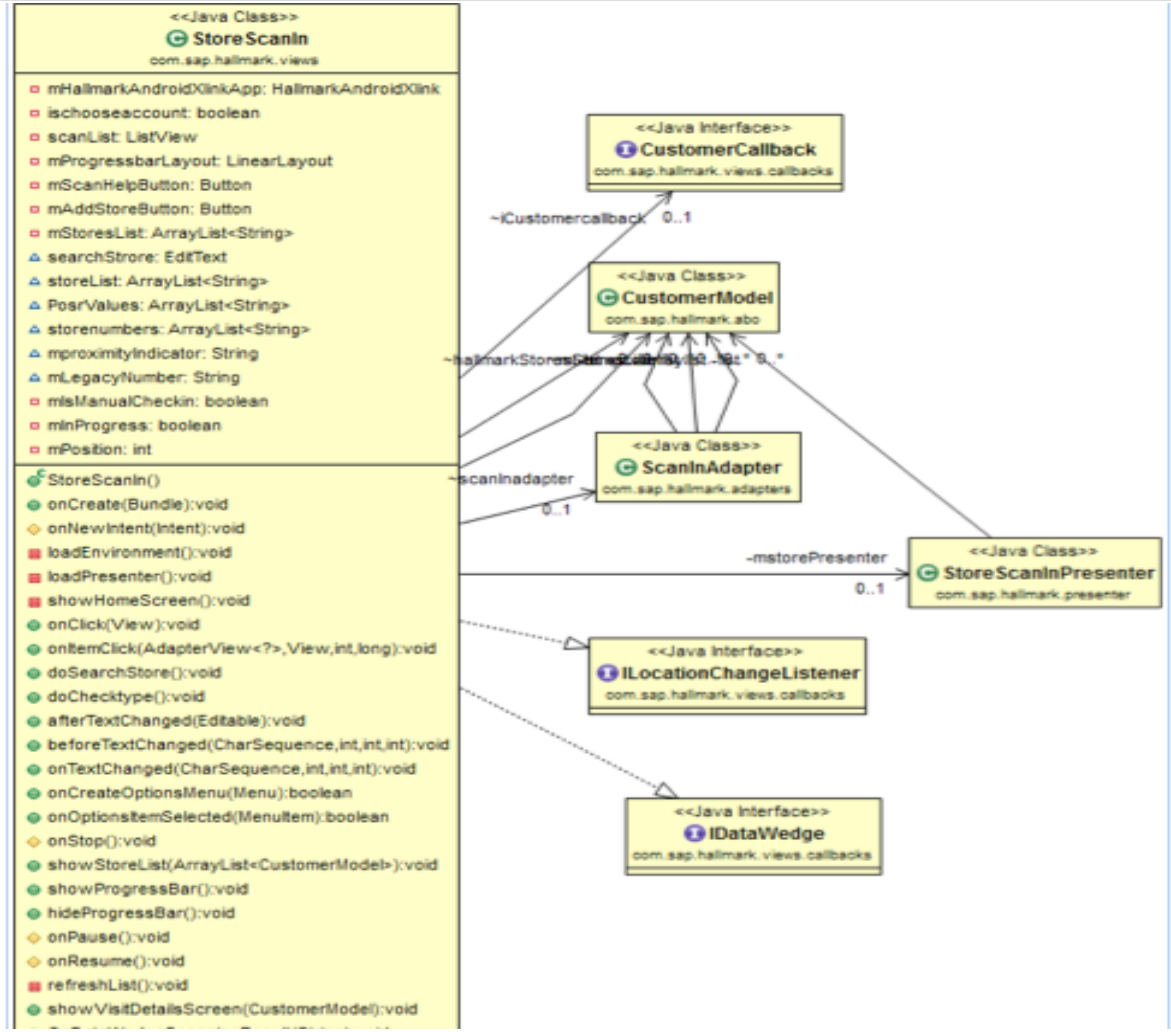
Ritika Mathur
Sep 14, 2015

Hallmark

Agenda

- General Program Structure
- Dependencies
- Packages
- Interfaces and Classes
- Store Visit GPS
- Location Utility
- Store Visit
- Store Verification
- Visit Details
- Checkout

Store Scan General Program Structure



Dependencies

The functionality has dependencies on the following activities:

- DataWedgeScanner
- CreditTallyProductListActivity
- SBTDiscardProductListActivity
- POSRProductListActivity.

Packages

The following packages are created to support this functionality:

- com.sap.hallmark.views
- com.sap.hallmark.adapters
- com.sap.hallmark.views.callbacks

Interfaces and Classes

Activity	StoreScanIn	Displays the list of stores available in user account
Class	StoreScanInPresenter	Controller class for StoreScanIn activity, it is mainly used to perform background operations
Interface	IStoreScanIn	Interface layer for StoreScanIn to send callback of next action
Class	ScanInAdapter	Adapter class to initialize and provide store lists views to activity class; it fetches the list of available stores from the database and returns views for each store item object
Interface	ILocationChangeListener	Listener for change in location
Interface	IStoreAsyncCall	Asynchronous call when the store list is generated successfully

Store Visit GPS

Application GPS update frequency

- 1 min & 10 m outside store
- 5 min & 50 m inside store

To detect GPS on device we have android notification service. So whether GPS is available or not will be notified by Android Notification.

Location Utility

LocationUtility class implements LocationListener

Static Instance

```
public static synchronized LocationUtility getLocationUtilityInstance() {
    if(mInstance == null) {
        mInstance = new LocationUtility();
    }
    return mInstance;
}
```

Multiple application can register for GPS update for a time

```
private LocationUtility() {
    mLocationListener = new ArrayList<ILocationChangeListener>(5);
}
```

Location Utility

Start GPS update

```
public void startGPSupdate(long minTime, long minDist, final Context context) {

    try {
        if(locationManager == null) {
            locationManager = (LocationManager) context
                .getSystemService(context.LOCATION_SERVICE);
        }

        // getting GPS status
        isGPSEnabled = locationManager
                .isProviderEnabled(LocationManager.GPS_PROVIDER);

        if (!isGPSEnabled) {
            // GPS is not enabled
            //showSettingsAlert();
        }

        // start requesting to GPS updates
        /*locationManager.requestLocationUpdates(
                LocationManager.GPS_PROVIDER,
                MIN_TIME_BW_UPDATES,
                MIN_DISTANCE_CHANGE_FOR_UPDATES, this);*/
        locationManager.requestLocationUpdates(
                LocationManager.GPS_PROVIDER,
                minTime,
                minDist, this);
```

Start GPS update

Calling StartGPS update

- On Create of Home Screen Activity
- While Check-in to store

To start receiving gps updates

* minTime - The time interval between two updates in milliseconds

* minDist - The distance between two updates in meter

* Context

```
LocationUtility
        .getLocationUtilityInstance()
        .startGPSupdate(
                IAppConstants.GPS.UPDATE_TIME_INSIDE_STORE,
                IAppConstants.GPS.UPDATE_DIST_INSIDE_STORE,
                TravelTimeActivity.this);
```

Stop GPS update

Calling Stop GPS update

- When no need of GPS update
- Called while exiting application

```
public void stopGPSupdate() {
    if(locationManager != null) {
        locationManager.removeUpdates(LocationUtility.this);
        locationManager = null;
    }
    if(mTimer != null){
        mTimer.cancel();
    }
    if(mLocationDeferTimer != null){
        mLocationDeferTimer.cancel();
    }
    clearAll();
    removeGPSNotification();
}
```

Get User Location Change

On Location Changed

```
@Override
public void onLocationChanged(final Location location) {
    // Update the new location in the application
    appUtilInstance.setLocation(location);

    if(mLocationListener != null) {
        for(int i=0; i<mLocationListener.size(); i++){
            mLocationListener.get(i).onReceivedNewLocation(location);
        }

    }
```

Set Location

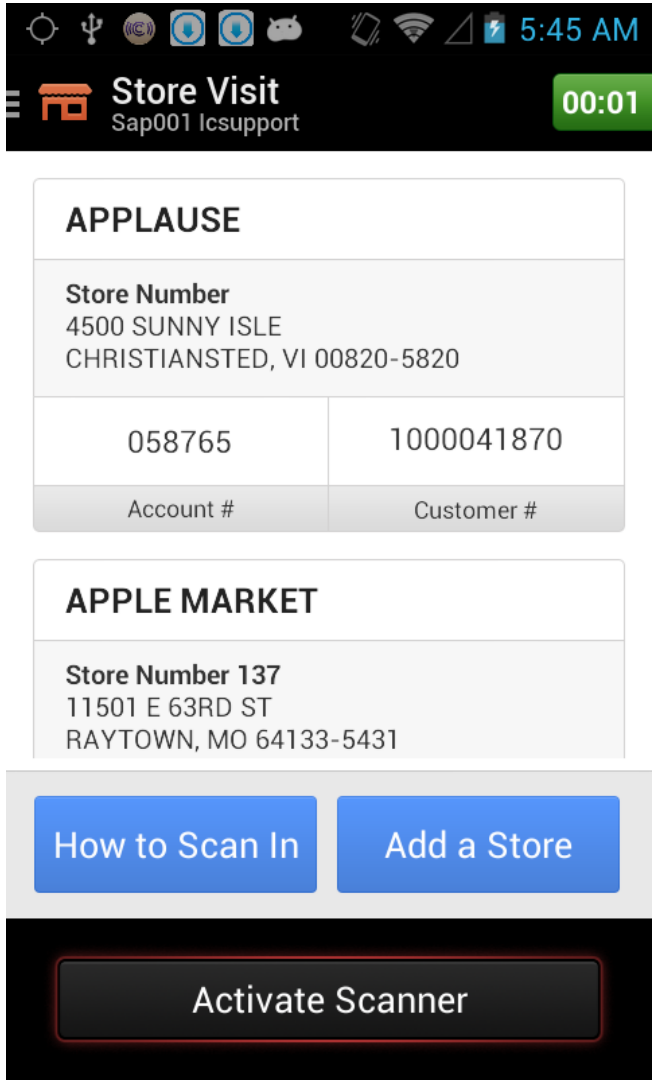
```
public void setLocation(Location location) {
    if (AppConfig.getAppConfigInstance(mContext)
            .getCustomerIdOfSuccessfulScanIn() == null
            || AppConfig.getAppConfigInstance(mContext)
            .getCustomerIdOfSuccessfulScanIn().isEmpty()) {
        // User not entered the store
        // So you have the freedom of updating whatever the location is
        mLocation = location;
        setLatitude(location == null ? null : BigDecimal
                .valueOf(location.getLatitude()));
        setLongitude(location == null ? null : BigDecimal
                .valueOf(location.getLongitude()));
    } else {
        // User is inside the store
        // If you are unable to fetch the location, just retain the checkin
        // (or) the latest data you have
        // If you are able to fetch the location, update it
        if (location != null) {
            mLocation = location;
            setLatitude(BigDecimal.valueOf(location.getLatitude()));
            setLongitude(BigDecimal.valueOf(location.getLongitude()));
        }
    }
}
```

Store Visit

Store Visit

List of stores is supplied by the server for that RM.

- Proximity distance should be from a global configuration MBO
- Use geo-coordinates from the Customer MBO
- By comparing stores & devices geo-location and proximity distance determine the proximity indicator (near, far, indeterminate)
- If the GPS coordinates are empty or zeroes in the customer MBO then it is indeterminate and manual check-in is needed
- The nearby accounts are listed on the top with ‘near by’
- The ‘far’ and ‘indeterminates’ are listed under ‘near by’s in a alpha sorted order



Store Visit

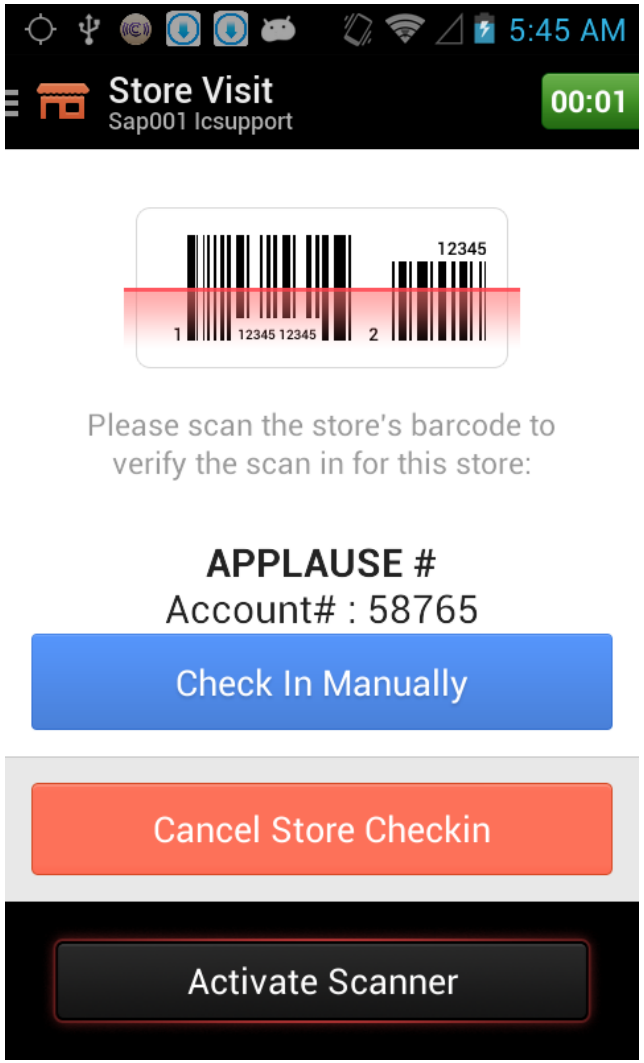
```
public void onClick(View view) {
    switch (view.getId()) {
    case R.id.bt_scan_help:
        Intent intent = new Intent(StoreScanIn.this, HelpActivity.class);
        startActivity(intent);
        break;
    case R.id.bt_add_store:
        Intent accountListIntent = new Intent(StoreScanIn.this,
                AccountsAddAccountActivity.class);
        startActivity(accountListIntent);

        break;
    default:
        super.onClick(view);
        break;
    }
}
```

Store Verification

This is next screen for any store which is not nearby. So user have to do Store Verification before it can enter to store. Here there are two options available

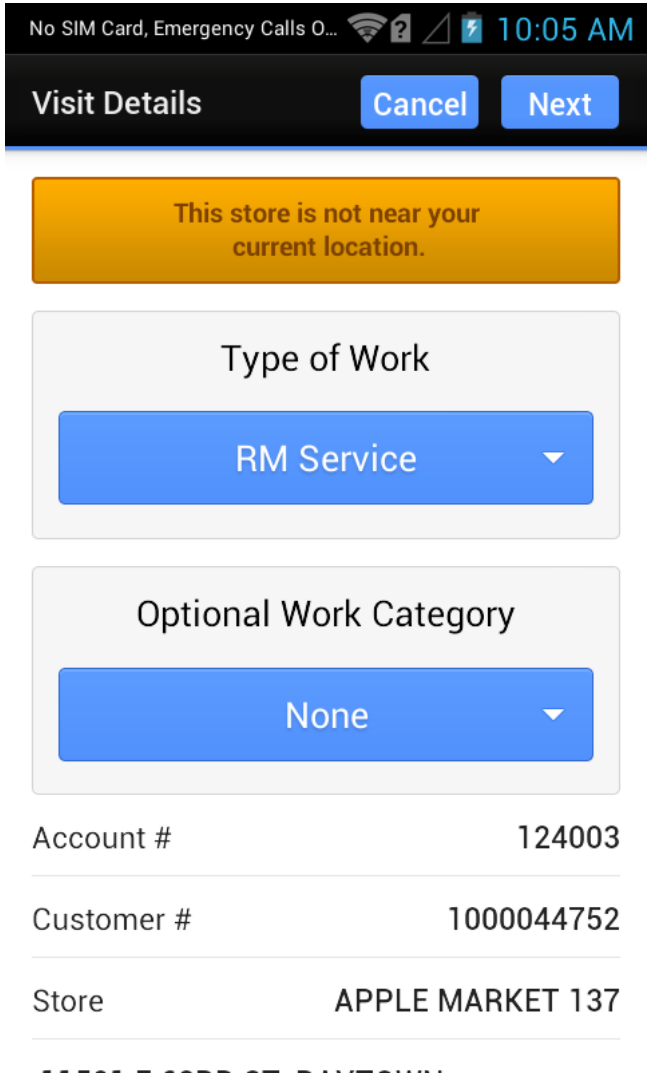
- Barcode scanning
- Manual Check - in



Visit Details

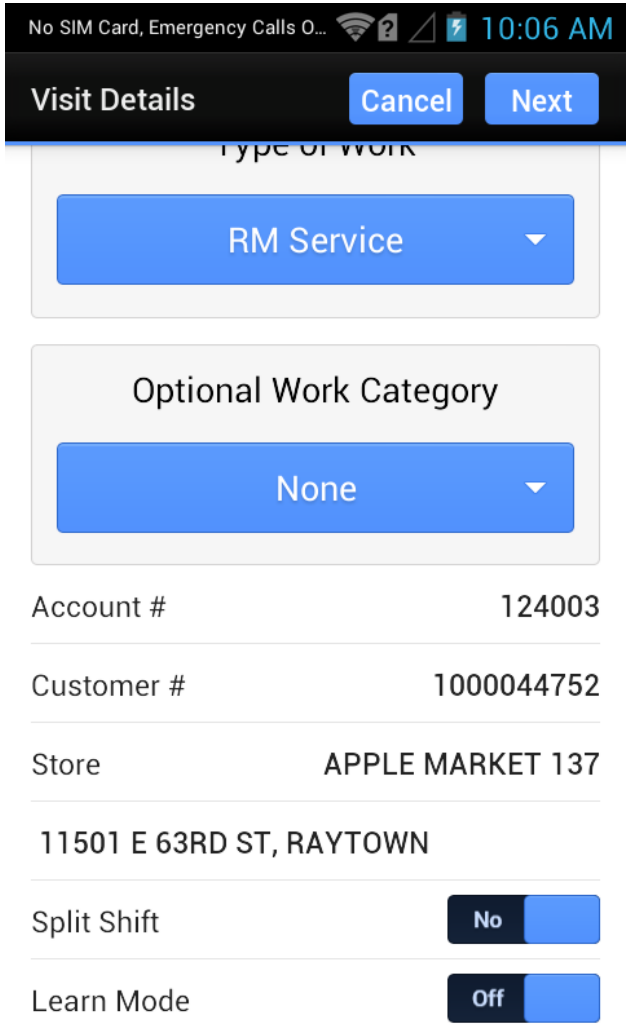
The user is presented with Visit Details after they have confirmed their visit is beginning, from GPS detection, or from manually scanning in.

- Display list of Type of Work options.
- Optional Work Category
- Toggle Split Shift on or off. Only displayed if the user is scanning into a store for the second time in the same calendar day.
- Toggle Learn Mode on and prompt the user with a warning.



Visit Details Continued

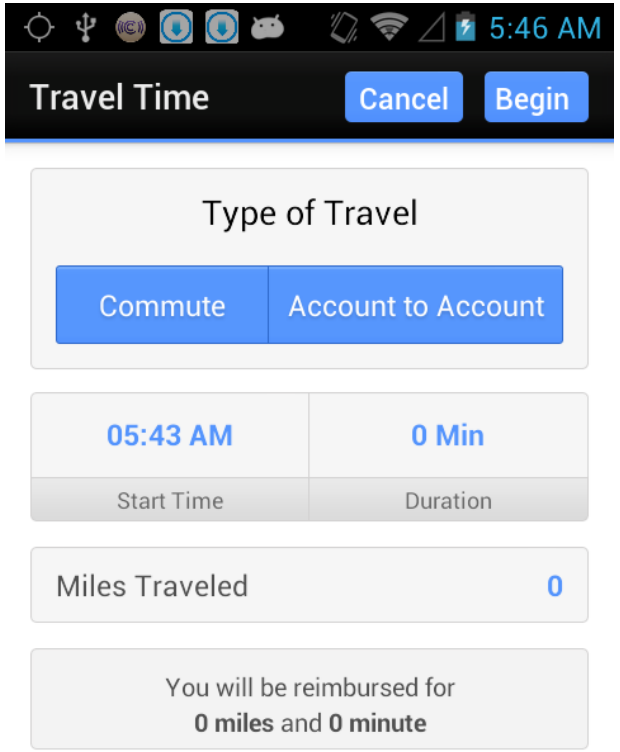
- Toggle Split Shift on or off. Only displayed if the user is scanning into a store for the second time in the same calendar day.
- Toggle Learn Mode on and prompt the user with a warning.



Visit Details: Travel Time

Travel Time is the second half of the Begin Visit flow, after the user has scanned into a store.

- By default no Type of Travel is selected, the user must pick one.
- Display time picker. See this screen for an example.
- Number pickers are used for both miles travelled and duration. The upper limit for mileage should be changeable and the prompt given should take into account that limit. (i.e. A limit of 99 would result in a prompt with two number pickers; a limit of 150 would result in a prompt with three, etc.)
- A small blurb informs the user of how much of the travel will be reimbursed.
- Confirms details entered and begins the visit.



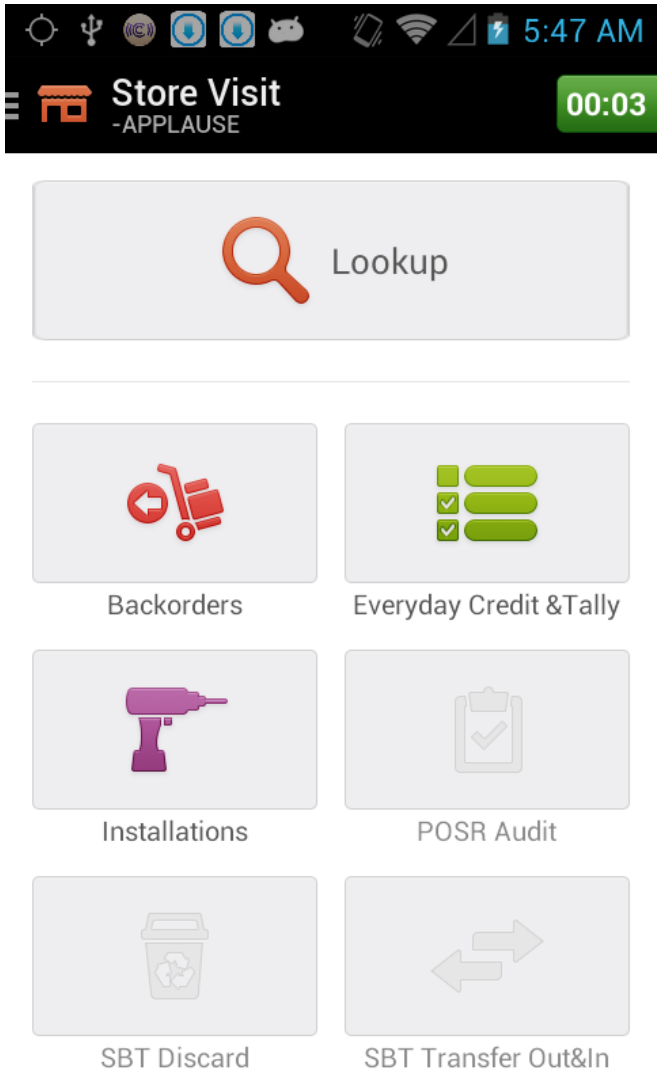
Store Visit : Scanned In

After the user has scanned into a store, they're able to access the Store Visit functions listed below.

These functions are arranged in a static grid

These items are arranged with Lookup as the most prominent. Other functions are listed alphabetically.

Functions are disabled depending on the account type; accounts can be either POSR/Non-POSR and either SBT/Non-SBT.



Store Visit : Functions

Here are the functions available, depending on the store. Non-POSR accounts may use POSR Audit but do not allow adding non-POSR products to the audit list.

	SBT POSR	SBT Non-POSR	Non-SBT POSR	Non-SBT Non-POSR
Lookup	x	x	x	x
Backorders	x	x	x	x
Everyday Credit & Tally			x	x
Installations	x	x	x	x
POSR Audit	x	o	x	
SBT Discard	x	x		
SBT Transfer In & Out	x	x		
Seasonal Credit & Tally			x	x
Shipments	x	x	x	x
Store Details	x	x	x	x

Store Scan View

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    loadEnvironment();
    loadPresenter();
    // choose account callback
    Intent intent = getIntent();

    if (intent.hasExtra(IAppConstants.CHOOSE_ACCOUNT_STORE_VISIT)) {
        ischooseaccount = intent.getBooleanExtra("addworktime", true);
    }
    // when the call comes from the help screen
    mIsManualCheckin = getIntent().getBooleanExtra(
            IAppConstants.EXTRA_IS_MANUAL_CHECKIN, false);
    showHomeScreen();
}
```

Store Listing

```
public void showStoreList(ArrayList<CustomerModel> list) {
    // int count = 0; // Variable to keep track of number of stores which
    // are
    // 'Nearby'
    mStores = list;
    mHallmarkAndroidXlinkApp.setAvailableStoresList(list);

    storeList = new ArrayList<String>();
    if (mStores != null && mStores.size() > 0) {
        scanList.setVisibility(View.VISIBLE);
        findViewById(R.id.empty_text).setVisibility(View.GONE);

        for (int i = 0; i < mStores.size(); i++) {
            storeList.add(mStores.get(i).getPosrIndicator());
        }

        Collections.sort(mStores, new Com
            String com.sap.hallmark.abo.CustomerModel.ge

            @Override
            public int compare(CustomerModel o1, CustomerModel o2) {

                String storeName1 = o1.getStoreName();
                String storeName2 = o2.getStoreName();

                // nearby accounts go on top
                boolean nearby1 = o1.getProximityIndicator()
                        .equalsIgnoreCase(IAppConstants.NEAR_BY);
                boolean nearby2 = o2.getProximityIndicator()
                        .equalsIgnoreCase(IAppConstants.NEAR_BY);
                if (nearby1 && !nearby2) {
                    return -1;
                } else if (nearby2 && !nearby1) {
                    return 1;
                } else {

                    if (storeName1 != null && storeName2 != null) {
                        int result = storeName1
                                .compareToIgnoreCase(storeName2);
                        if (result == 0) {
                            // both accounts have the same name; look at the
                            // store number
                            String storeNumber1 = o1.getStoreNumber2();
                            String storeNumber2 = o2.getStoreNumber2();
                            if (storeNumber1 != null
                                    && storeNumber2 != null) {
                                result = storeNumber1
                                        .compareToIgnoreCase(storeNumber2);
                            } else if (storeNumber1 != null) {
                                // empty store numbers are last
                                result = -1;
                            } else if (storeNumber2 != null) {
                                // empty store numbers are last
                                result = 1;
                            }
                        }
                        return result;

                    } else if (storeName1 != null)
                        return -1;
                    else
                        return 1;
```

Scan Barcode

```
public void OnDataWedgeScanningResult(String result, String labelType) {

    if (mStores != null) {
        boolean isStoreFoundInlist = false;
        AppConfig appconfig = AppConfig
                .getAppConfigInstance(getApplicationContext());
        int upcCodeLength = result.length() - 1;
        String legacyNumberOnScan = result.substring(0, upcCodeLength);
        appconfig.setSelectedStoreLegacyNumber(legacyNumberOnScan);

        for (int i = 0; i < mStores.size(); i++) {
            String legacyNumber = mStores.get(i).getLegacyNumber();
            if ((legacyNumber != null) && (legacyNumberOnScan != null)
                    && (legacyNumber.equalsIgnoreCase(legacyNumberOnScan))) {
                isStoreFoundInlist = true;

                break;
            }
        }
        if (isStoreFoundInlist) {
            if (legacyNumberOnScan != null) {
                mstorePresenter.getProximityIndicator(legacyNumberOnScan);
                mstorePresenter.setStoreCheckInType(
                        ABOConstants.CHECKIN_TYPE_SCAN, legacyNumberOnScan);
            } else {
                Toast.makeText(this, "Invalid Store!!!", Toast.LENGTH_SHORT)
                        .show();
                finish();
            }
        } else {
            Toast.makeText(this, "Store not available in the list!!!",
                    Toast.LENGTH_SHORT).show();
        }
    } else {
        Toast.makeText(this, "Store not available in the list!!!",
                Toast.LENGTH_SHORT).show();
    }

}
```

Store Listing

```
public void getStoreList() {
    StoreScanABO storescanMbo = new StoreScanABO(mContext);
    storescanMbo.setCallback(this);
    storescanMbo.execute();

}
```

Thank you

Contact information:

Ritika Mathur (C5227190)
Bangalore , India