BINUS UNIVERSITY INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	Surname	Given Names	Student ID Number
1.	Senjaya	Monique	2440061285

Course Code : COMP 6699 **Course Name** : Object Oriented Programming

Class : L2BC **Name of Lecturer(s)** : Jude Joseph Lamug Martinez

Major : Computer Science

Title of Assignment: Pocketz (Grocery Inventory Management)

Type of Assignment : Final Project

Submission Pattern

Due Date : 22/06/2021 **Submission Date** : 22/06/2021

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.

4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

1. Monique Senjaya

Table of Contents

Cover letter 1

Table of Contents 3

A. Description

I. Introduction 4

II. The function of the program 4

B. Solution Design Plan

I. Parts of the program 5

II. Function of each parts of the program 8

C. Implementation

I. Data dictionary (item.db) 9

II. Classes diagram (UML) 10

III. Explanation of classes and functionalities 11

D. Proof of Working Product 25

“Pocketz”

Name: Monique Senjaya

ID: 2440061285

A. Description

I. Introduction

When the final project was first introduced, I started to brainstorm and search on what I wanted to make. It was really difficult to choose what I wanted to make, but in the end I figured that it would be a lot better if I tried to solve a day-to-day problem that I seem to be having. Then, I realized that one minor but tedious problem I had was having to check my groceries stock before going to shop. I thought that it would be so much more convenient if I had the information already in my phone. This way, I can just check what I should restock and what’s still in-stock. As this is my first time doing a proper java project, I plan to do more research on how the java library works and also how to make an android app. I am really looking forward to this project.

After a lot of research and trials, I officially begin this project on the 24th of May 2021. I chose to work on this project with Android Studios and I will commit and push all progress to my GitHub account, under this repository https://github.com/moniquesenjaya/pocketz.

II. The function of the program

The purpose of this program is to allow users to easily access and find out their groceries stocks remotely through the use of their smartphones. Not only that, this program can also be used as a reminder for groceries that are close to expiry. This will reduce waste of food and drinks and also make shopping more time efficient.

Users will have to download and install the application to their smartphones. Once they launch the app, a database will be made for them. They will have to input the item details to the form in the app (item name, quantity, category, expiry date and storage). Each of this data will be inserted to the database (sqlite) and whenever an item is used, users can swipe it off the screen to update the inventory. If the user wants to check what items will go off soon, they can go to the near expiry section and check. If they want to go grocery shopping, they can check the grocery list and see which item categories are out of stock. Other than that, if they forgot

where they put a certain item they can just check where it is stored from the app too, this way they won't misplace it and forget that they've bought the item.

Specifications regarding the project includes:

- Input of the program:
  - Item details (item name, quantity, category, expiry date and storage)
  - Swipe item off the list when used
- Output of the program:
  - List of grocery at stock
  - List of items near expiry
  - List of items out of stock
- Java Util used:
  - java.util.ArrayList
  - java.util.Arrays
  - java.util.List
  - java.util.Date
  - java.util.Calendar

B. Solution Design Plan

I. Parts of the program

In this project, I will be using android and the GUI will be designed using .xml. The screen will consist of 4 pages that are important to help the user navigate the workings of the application. The list of the sections are:

- Home page
- Item entry form page
- Grocery list page
- Expiry items list page

Each page will be accessible through the click of a button in the home page. The figure below will further visualize this.

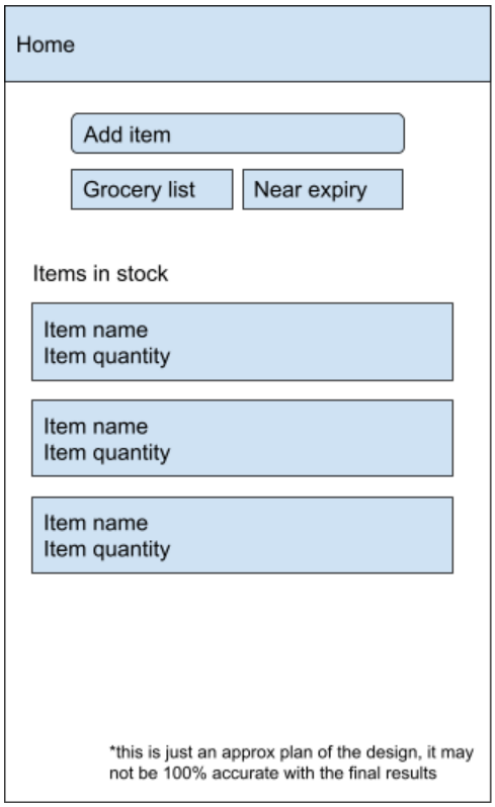


Figure 1: Home page

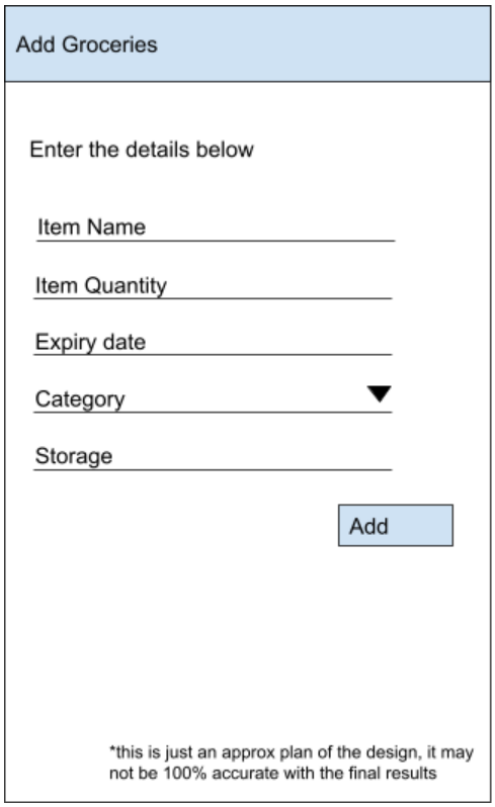


Figure 2: Add item page

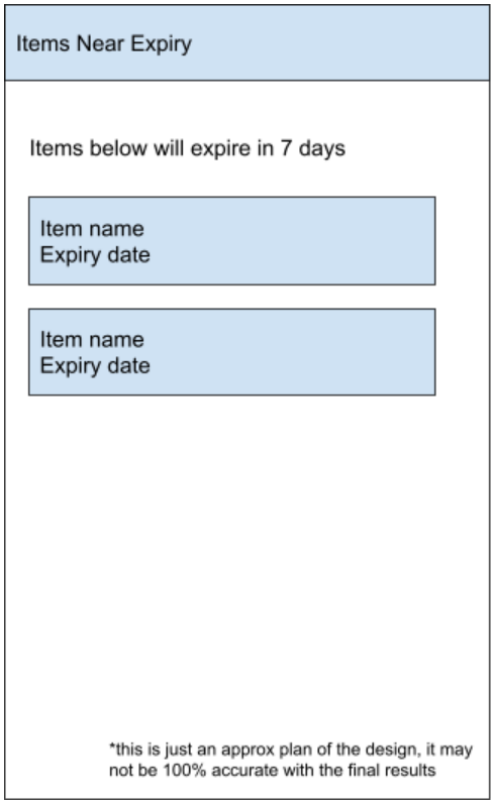


Figure 3: Items near expiry page

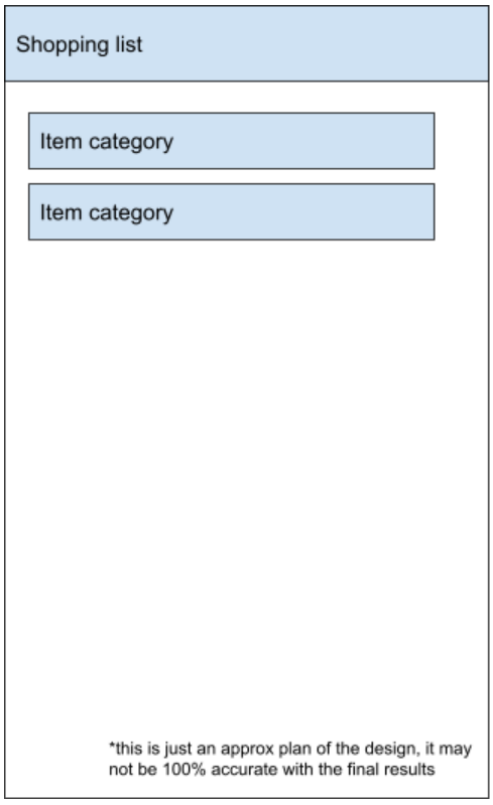


Figure 4: Shopping list page

II. Function of each parts of the program

Home page

This part of the program functions as a tool for the user to navigate through the different pages. It will consist of 3 buttons and a recycler view. The three buttons are to navigate to add item page, shopping list page, and expiry page. The recycler view is used to show the inventory of the groceries that are in stock. In this page, the user can swipe off the items in the recycler view everytime an item is used, the database will then be updated. The view will only show items in stock, so items with 0 quantity will not be included.

Item entry form page

This part of the program functions as a tool for the user to input the details of the items that were just bought. This is where the item name, quantity, expiry date, category and storage are being typed in and stored to the database. It allows the user to easily insert data into the database without having to access the database. In this page, I will use various entry tools such as EditTexts and Spinner (dropdown menu) to make a better user experience.

Grocery list page

This part of the program will contain a recycler view that will show the items that are out of stock. It will display the category that has a quantity of 0. The items in the grocery list will automatically be deleted when new items of that category are being inserted to the database.

Expiry items list page

This part of the program will also contain a recycler view that will show the items that will expire in a week (7 days). When a user buys a lot of groceries, it is common for them to forget what they bought and it might lead to a waste of food or drinks. Having this feature helps remind them of food that will go off soon and remind them to consume it within the week.

C. Implementation

I. Data dictionary (item.db)

ITEM_TABLE

Field Name	Data Type	Description
ITEM_NAME	text	Shows the item name
ITEM_QTY	int	Shows the amount of item
ITEM_EXP	text	Shows the expiry date of the item (YYYY/MM/DD)
ITEM_CATEGORY	text	Shows the category of the item provided in the drop down list of the form
ITEM_STORAGE	text	Shows where the item is stored

II. Classes Diagram (UML)

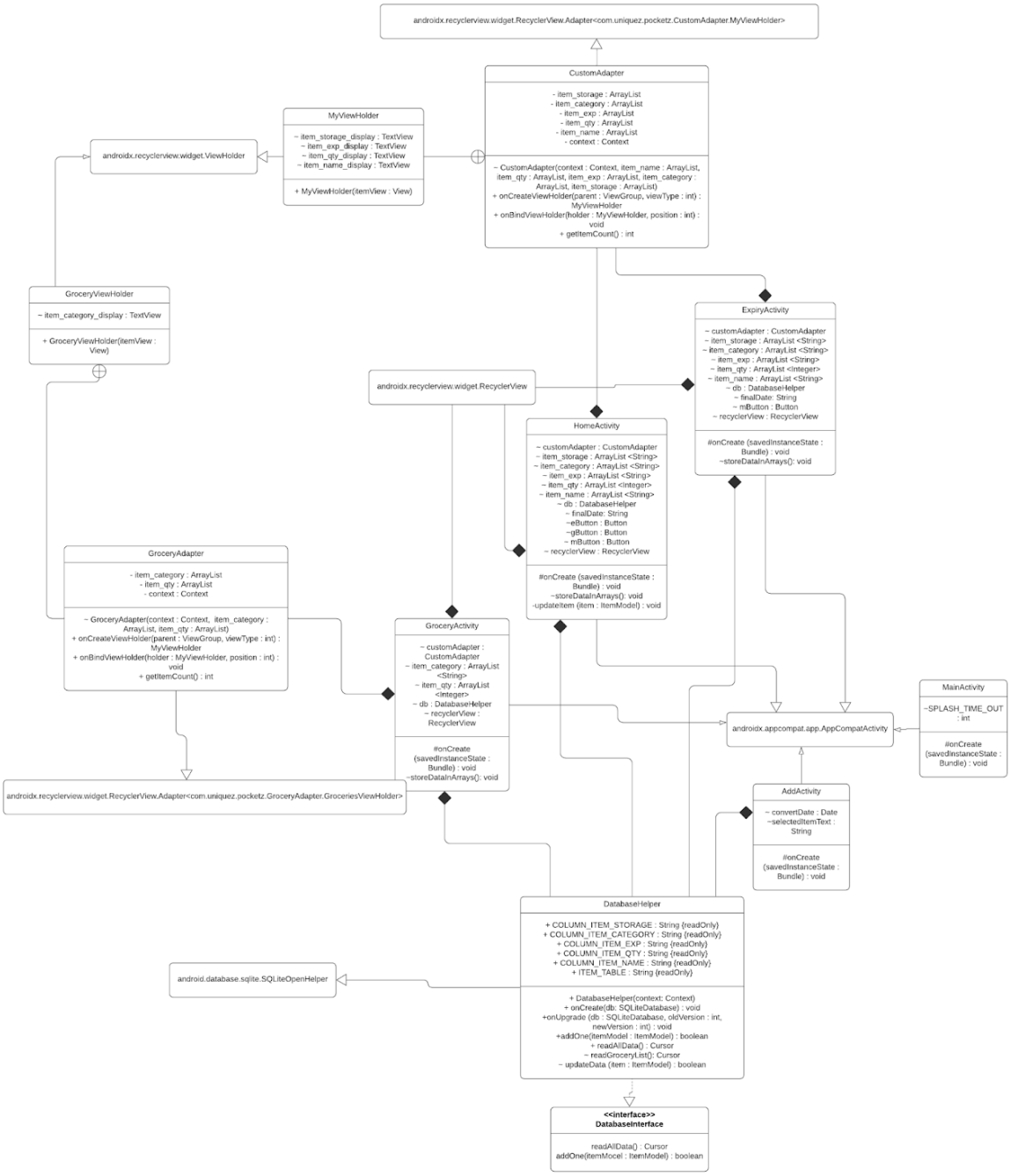


Figure 5: UML of the classes

III. Explanation of classes and functionalities

ItemModel.java

```java
package com.uniquez.pocketz;

import java.util.Date;

public class ItemModel {
    private String name;
    private int qty;
    private Date expDate;
    private String category;
    private String storage;

    //Non-param constructor
    public ItemModel() {
    }

    //Param constructor without exp date
    public ItemModel(String name, int qty, String category, String storage) {
        this.name = name;
        this.qty = qty;
        this.category = category;
        this.storage = storage;
    }

    //Param constructor
    public ItemModel(String name, int qty, Date expDate, String category, String storage) {
        this.name = name;
        this.qty = qty;
        this.expDate = expDate;
        this.category = category;
        this.storage = storage;
    }
```

Figure 6: ItemModel class

This is the custom class I made to store item objects. It contains 5 attributes. Name, quantity, expiry date, category and storage. I made 3 constructors. One without parameters, one with all the parameters, and the last one a constructor with parameters but without an expiry date. This is because not all groceries have exp dates on them.

```java
//getter and setters
public String getName() { return name; }

public void setName(String name) { this.name = name; }

public int getQty() { return qty; }

public void setQty(int qty) { this.qty = qty; }

public Date getExpDate() { return expDate; }

public void setExpDate(Date expDate) { this.expDate = expDate; }

public String getCategory() { return category; }

public void setCategory(String category) { this.category = category; }

public String getStorage() { return storage; }

public void setStorage(String storage) { this.storage = storage; }

//to string method
@Override
public String toString() {
    return "ItemModel{" +
            "name='" + name + '\'' +
            ", qty=" + qty +
            ", expDate=" + expDate +
            ", category='" + category + '\'' +
            ", storage='" + storage + '\'' +
            '}';
}
```

Figure 7: ItemModel getter and setter

Aside from constructors, I made getters and setters. I also included the toString method and overridden it just in case I needed it for debugging.

HomeActivity.java

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_home);

    //Button to go to add activity
    mButton = findViewById(R.id.addButton);
    mButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view)
        {
            Intent addIntent = new Intent( packageContext: HomeActivity.this, AddActivity.class);
            startActivity(addIntent);
        }
    });

    //Button to go to grocery activity
    gButton = findViewById(R.id.groceryButton);
    gButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view)
        {
            Intent addIntent = new Intent( packageContext: HomeActivity.this, GroceryActivity.class);
            startActivity(addIntent);
        }
    });

    //Button to go to expiry activity
    eButton = findViewById(R.id.expiryButton);
    eButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent expiryIntent = new Intent( packageContext: HomeActivity.this, ExpiryActivity.class);
            startActivity(expiryIntent);
        }
    });
```

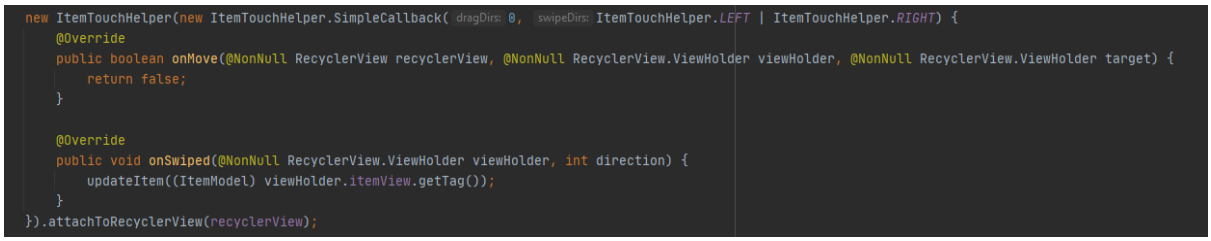
Figure 8: onCreate method

The first function in this class is the onCreate method. This is a method that is overridden from AppCompatActivity. It runs every time the activity is created. Here, I added 3 onClickListener for the 3 buttons in the Home Page. Inside the listener, I override the onClick function and made a new intent object to navigate the activities.

```
recyclerView = findViewById(R.id.recyclerView);

db = new DatabaseHelper( context: HomeActivity.this);
item_name = new ArrayList<>();
item_qty = new ArrayList<>();
item_exp = new ArrayList<>();
item_category = new ArrayList<>();
item_storage = new ArrayList<>();

Log.e( tag: "databasecheck",  msg: "outside try");

try {
    Log.e( tag: "databasecheck",  msg: "does it");
    storeDataInArrays();
    Log.e( tag: "databasecheck", item_category.toString());
} catch (ParseException e) {
    e.printStackTrace();
}

customAdapter = new CustomAdapter( context: HomeActivity.this, item_name, item_qty, item_exp, item_category, item_storage);
recyclerView.setAdapter(customAdapter);
recyclerView.setLayoutManager(new LinearLayoutManager( context: HomeActivity.this));

if (customAdapter.getItemCount() == 0){
    TextView instruct = findViewById(R.id.no_data);
    instruct.setVisibility(View.VISIBLE);
}
```

Figure 9: Attaching the recyclerView

Aside from setting listeners for the buttons, I also included code to bind data to the recyclerview. This aims to show all the items in stock. To do this, I referenced the recyclerView, made a new database object, and also 5 array lists for the item details. DatabaseHelper class will be further discussed below. The storeDataInArrays() is a function that reads the database and inserts the data from it to the array lists made. A CustomAdapter object is then made with the array lists as the arguments. Then, the object will be passed to the recycler view as the adapter to bind the data. The if block checks if the adapter count is 0 (no data) it will show a textview that data is not available. This aims to make the UX better.



```
new ItemTouchHelper(new ItemTouchHelper.SimpleCallback( dragDirs: 0,  swipeDirs: ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT) {
    @Override
    public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull RecyclerView.ViewHolder viewHolder, @NonNull RecyclerView.ViewHolder target) {
        return false;
    }

    @Override
    public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {
        updateItem((ItemModel) viewHolder.itemView.getTag());
    }
}).attachToRecyclerView(recyclerView);
```

Figure 10: Setting the rows to be updated on swiped

The last part of the onCreate() methods to allow the users to swipe the items left or right whenever the item is used. This is to reduce the item quantity. ItemTouchHelper object is made

and attached to the recycler view. I override the onSwipped() method and set it so that it updates the database and the recyclerView.

```
void storeDataInArrays() throws ParseException {
    Cursor cursor = db.readAllData();
    if(cursor.getCount() == 0){
        Toast.makeText( context: this,  text: "No data", Toast.LENGTH_SHORT).show();
    }else{
        while (cursor.moveToNext()){

            if(cursor.getInt( columnIndex: 1) != 0){
                if (cursor.getString( columnIndex: 2).equals("0")){
                    item_exp.add("0");
                }else{
                    SimpleDateFormat formatter = new SimpleDateFormat( pattern: "EEE MMM dd HH:mm:ss zzzz yyyy");
                    SimpleDateFormat formatterFinal = new SimpleDateFormat( pattern: "yyyy/MM/dd");
                    try {
                        Date date = formatter.parse(cursor.getString( columnIndex: 2));
                        Log.e( tag: "datecheck", date.toString());
                        finalDate = formatterFinal.format(date);
                        Log.e( tag: "datecheck", finalDate);
                    } catch (ParseException e) {
                        e.printStackTrace();
                    }
                }
                item_name.add(cursor.getString( columnIndex: 0));
                item_qty.add(cursor.getInt( columnIndex: 1));
                item_exp.add(finalDate);
                item_category.add(cursor.getString( columnIndex: 3));
                item_storage.add(cursor.getString( columnIndex: 4));
            }
        }
    }
    Log.e( tag: "databasecheck", item_name.toString());
}
```

Figure 11: storeDataInArrays() method

Aside from the onCreate method, HomeActivity.java contains this function that was used to insert database data into the arrays for the recyclerview. Here, a cursor object is made to store the return value of readAllData(). This function will be discussed when the DatabaseHelper class is discussed. If the cursor does not contain data, it doesn't run the while loop, else, it will loop through the cursor. Here, the data will be read and inserted to the array. The date will be first converted to the format to make it uniform. To do this, I used the try and catch to catch any errors. As the input of expiry date is not required, if there is no data, I insert 0 as the data.

```
private void updateItem(ItemModel item){
    boolean result = db.updateData(item);
    if (result){
        Toast.makeText( context: this,  text: "Successfully Updated", Toast.LENGTH_SHORT).show();
        recreate();
    }else{
        Toast.makeText( context: this,  text: "Failed", Toast.LENGTH_SHORT).show();
    }

}
```

Figure 12: updateItem() function

This is the last method in the class. This is to update the item in case of swiping the data. Result is made to store the return value of updateData() which will be discussed in the DatabaseHelper class.

AddActivity.java

```
//Making Array List of category
final List<String> categoryList = new ArrayList<>(Arrays.asList(category));

// Initializing an ArrayAdapter
final ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<~>(
        context: this,R.layout.spinner_item,categoryList){
    @Override
    public boolean isEnabled(int position){
        // Disable the first item from Spinner
        // First item will be use for hint
        return position != 0;
    }
    @Override
    public View getDropDownView(int position, View convertView,
                                ViewGroup parent) {
        View view = super.getDropDownView(position, convertView, parent);
        TextView tv = (TextView) view;
        if(position == 0){
            // Set the hint text color gray
            tv.setTextColor(Color.GRAY);
        }
        else {
            tv.setTextColor(Color.BLACK);
        }
        return view;
    }
};
spinnerArrayAdapter.setDropDownViewResource(R.layout.spinner_item);
spinner.setAdapter(spinnerArrayAdapter);
```

Figure 13: Creating a spinner

For the adding items page, I made a spinner for the item category. This is to reduce errors and also make input more efficient. Here, I made an array list containing the pre-set value that I declared. Then, I initialized the array adapter where I override 2 methods. The getDropDownView method is used to control the display of the data. If the position is 0, the color is gray if not, black. Then, the adapter is passed to the spinner into the spinner.

```
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        // If user change the default selection
        // First item is disable and it is used for hint
        if(position > 0){
            // Notify the selected item text
            selectedItemText = (String) parent.getItemAtPosition(position);
        }
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {

    }
});
```

Figure 14: Storing the selected data from spinner

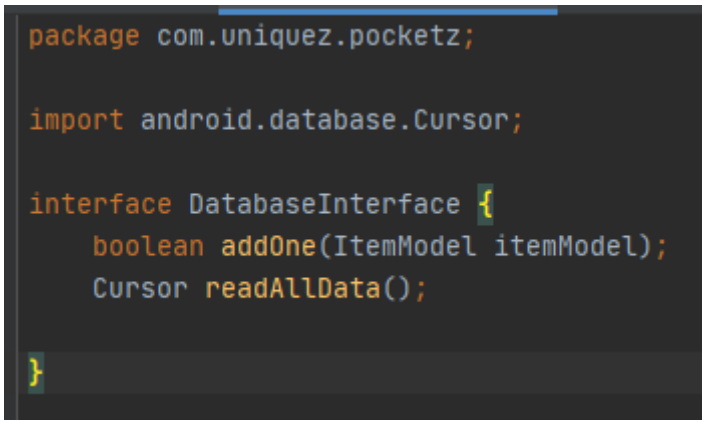
This block of code is used to store the selected item to a variable. Here, the onItemSelected() method is being overridden. Whenever the user changes the selection, it will store it to the selectedItemText variable.

```
addButton.setOnClickListener(new View.OnClickListener(){
    @SuppressLint("SimpleDateFormat")
    @Override
    public void onClick(View v){
        ItemModel itemModel = null;
        try{
            //check if exp date is avail
            if(expiryDate.getText() == null){
                itemModel = new ItemModel(itemName.getText().toString(), Integer.parseInt(itemQty.getText().toString()), selectedItemText, storageDetails.getText().toString());
            }else{
                //try to convert the date
                try {
                    convertDate = new SimpleDateFormat( pattern: "yyyy/MM/dd").parse(expiryDate.getText().toString());
                } catch (ParseException e) {
                    e.printStackTrace();
                }
                itemModel = new ItemModel(itemName.getText().toString(), Integer.parseInt(itemQty.getText().toString()), convertDate , selectedItemText, storageDetails.getText().toST
            }
        } catch (Exception e){
            Toast.makeText( context: AddActivity.this,  text: "Error in making item", Toast.LENGTH_SHORT).show();
        }
        try{
            DatabaseHelper databaseHelper = new DatabaseHelper( context: AddActivity.this);
            boolean success = databaseHelper.addOne(itemModel);

            if (success){
                Toast.makeText( context: AddActivity.this,  text: "Data added to the database", Toast.LENGTH_SHORT).show();
                Intent addIntent = new Intent( packageContext: AddActivity.this, HomeActivity.class);
                startActivity(addIntent);
            }
        }catch(Exception e){
            Toast.makeText( context: AddActivity.this,  text: "Please fill in the required fields", Toast.LENGTH_SHORT).show();
        }

    }
```

Figure 15: Adding data to database

This method is used to insert data to the database. Whenever the add button is pressed, it will check if the expiry date is null. If it is, it will use the constructor where exp date is not needed. If it isn't null, it will convert the date to the YYYY/MM/DD format to make it uniform. It will catch errors and make a toast if it fails to make the object. After making the new object, it will try to insert it to the database. It tries to insert using the addOne method from DatabaseHelper. If it fails, it will provide an error message.

DatabaseInterface.java



```
package com.uniquez.pocketz;

import android.database.Cursor;

interface DatabaseInterface {
    boolean addOne(ItemModel itemModel);
    Cursor readAllData();

}
```

Figure 16: Interface for database helper

Since all database will need insert and select methods, I made this interface to allow my helper class implement these two methods.

DatabaseHelper.java

```
public DatabaseHelper(@Nullable Context context) { super(context, name: "item.db", factory: null, version: 1); }

//first time a database is accessed
@Override
public void onCreate(SQLiteDatabase db) {
    String createTableStatement = "CREATE TABLE " +
            ITEM_TABLE + " (" +
            COLUMN_ITEM_NAME + " TEXT, " +
            COLUMN_ITEM_QTY + " INT, " +
            COLUMN_ITEM_EXP + " TEXT, " +
            COLUMN_ITEM_CATEGORY + " TEXT, " +
            COLUMN_ITEM_STORAGE + " TEXT)";

    db.execSQL(createTableStatement);
}
```

Figure 17: DatabaseHelper onCreate method

This DatabaseHelper will create a new database named “item.db” when it is being run the first time. When it is accessed for the first time, it will run the onCreate function which will create a table with details as stated in the data dictionary given above. This onCreate is overridden from the SQLiteOpenHelper as it inherits from that class.

```
//called when there are data that will be inserted to DB
@Override
public boolean addOne (ItemModel itemModel){

    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();

    cv.put(COLUMN_ITEM_NAME, itemModel.getName());
    cv.put(COLUMN_ITEM_QTY, itemModel.getQty());
    try{
        cv.put(COLUMN_ITEM_EXP, itemModel.getExpDate().toString());
    }catch (Exception e){
        cv.put(COLUMN_ITEM_EXP, "0");
    }

    cv.put(COLUMN_ITEM_CATEGORY, itemModel.getCategory());
    cv.put(COLUMN_ITEM_STORAGE, itemModel.getStorage());

    long insert = db.insert(ITEM_TABLE,  nullColumnHack: null, cv);

    if (insert == -1){
        return false;
    }
    else{
        return true;
    }
}
```

Figure 18: addOne() method from the interface

This function is used to insert data to the table. It initializes the db to have to write for the database. It also creates cv, which stores a list of key value pairs. ItemModel object will be passed to this function and all the values will be stored in cv. The insert function is then used to insert the values to the table. It will return -1 if not succeed.

```
@Override
public Cursor readAllData(){
    String query = "SELECT * FROM " + ITEM_TABLE;
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = null;
    if (db != null){
        cursor = db.rawQuery(query, selectionArgs: null);
    }
    return cursor;
}
```

Figure 19: readAllData() method for select query

This method is used to get all the data from the item table without any filter. A query is used to select * (all data) from the table. This time db is set so that it allows reading of data. It will return a cursor that contains the data from the table.

```
boolean updateData(ItemModel item){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();
    cv.put(COLUMN_ITEM_QTY, (item.getQty()-1));
    long result = db.update(ITEM_TABLE,
            cv,
            whereClause: "ITEM_NAME=? AND ITEM_CATEGORY=? AND ITEM_STORAGE=?",
            new String[] {item.getName(), item.getCategory(), item.getStorage()});
    return result != -1;
}
```

Figure 20: updateData() method for the swipe action

This method is used to subtract the quantity of the item from the database. The db is set to writable and a cv is declared. It will store the current item quantity - 1. Update method will be used here to update the item table with the cv value where the item name, category and storage is equal to the ItemModel's data. If it doesn't succeed, it will return false.

```
Cursor readGroceryList(){
    String query = "SELECT " + COLUMN_ITEM_CATEGORY +
            " , SUM(" + COLUMN_ITEM_QTY + ")" +
            " FROM " + ITEM_TABLE + " GROUP BY " +
            COLUMN_ITEM_CATEGORY;
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = null;
    if (db != null){
        cursor = db.rawQuery(query, selectionArgs: null);
    }
    return cursor;
}
```

Figure 21: readGroceryList() method

This last method in the DatabaseHelper class will be used to display the shopping list. Here, instead of selecting all the data, the query selects only the category and sum of qty. It will be displayed in groups of the category. The rest of the method follows the steps of the readAllData() method.

CustomAdapter.java

```
private Context context;
private ArrayList item_name, item_qty, item_exp, item_category, item_storage;

CustomAdapter(Context context, ArrayList item_name, ArrayList item_qty, ArrayList item_exp, ArrayList item_category, ArrayList item_storage){
    this.context = context;
    this.item_name = item_name;
    this.item_qty = item_qty;
    this.item_exp = item_exp;
    this.item_category = item_category;
    this.item_storage = item_storage;
}

@NonNull
@Override
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    LayoutInflater inflater = LayoutInflater.from(context);
    View view = inflater.inflate(R.layout.recycler_row, parent, attachToRoot: false);
    return new MyViewHolder(view);
}

@Override
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
    holder.item_name_display.setText(String.valueOf(item_name.get(position)));
    holder.item_qty_display.setText(String.valueOf(item_qty.get(position)));
    if (String.valueOf(item_exp.get(position)).equals("0")){
        holder.item_exp_display.setText("");
    }else{
        holder.item_exp_display.setText(String.valueOf(item_exp.get(position)));
    }
    holder.item_storage_display.setText(String.valueOf(item_storage.get(position)));
    holder.itemView.setTag(new ItemModel(String.valueOf(item_name.get(position)),
            (Integer) item_qty.get(position), String.valueOf(item_category.get(position)),
            String.valueOf(item_storage.get(position))));
}
```


Figure 22: CustomAdapter() for recycler view

RecyclerView includes a new kind of Adapter. It requires a ViewHolder for handling Views. We will have to override two main methods first one to inflate the view and its viewholder and another one to bind the data to the view. The main good thing in this is that the first method is called only when we really need to create a new view.

```
}

@Override
public int getItemCount() { return item_name.size(); }

public class MyViewHolder extends RecyclerView.ViewHolder{

    TextView item_name_display, item_qty_display, item_exp_display, item_storage_display;

    public MyViewHolder(@NonNull View itemView) {
        super(itemView);
        item_name_display = itemView.findViewById(R.id.itemNameDisplay);
        item_qty_display = itemView.findViewById(R.id.itemQtyDisplay);
        item_exp_display = itemView.findViewById(R.id.itemExpDisplay);
        item_storage_display = itemView.findViewById(R.id.itemStorageDisplay);

    }
}
```

Figure 23: ViewHolder class

ViewHolder class is used to store the reference of the View's for one entry in the RecyclerView. A ViewHolder is a static inner class in our Adapter which holds references to the relevant view's. By using these references our code can avoid the time consuming findViewById() method to update the widgets with new data.

ExpiryActivity.java and GroceryActivity.java

```java
@RequiresApi(api = Build.VERSION_CODES.O)
void storeDataInArrays() throws ParseException {
    Cursor cursor = db.readAllData();
    if(cursor.getCount() == 0){
        Toast.makeText( context: this, text: "No data", Toast.LENGTH_SHORT).show();
    }else{
        while (cursor.moveToNext()){
            Calendar calendar = Calendar.getInstance();
            SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "yyyy/MM/dd");
            Object date = dateFormat.format(calendar.getTime());
            if(cursor.getInt( columnIndex: 1) != 0){
                if (cursor.getString( columnIndex: 2).equals("0")){
                    item_exp.add("0");
                }else{
                    SimpleDateFormat formatter = new SimpleDateFormat( pattern: "EEE MMM dd HH:mm:ss zzzz yyyy");
                    SimpleDateFormat formatterFinal = new SimpleDateFormat( pattern: "yyyy/MM/dd");
                    try {
                        Date expDate = formatter.parse(cursor.getString( columnIndex: 2));
                        finalDate = formatterFinal.format(expDate);
                    } catch (ParseException e) {
                        e.printStackTrace();
                    }
                }
                final DateTimeFormatter formatters = DateTimeFormatter.ofPattern("yyyy/MM/dd");
                final LocalDate firstDate = LocalDate.parse(date.toString(), formatters);
                final LocalDate secondDate = LocalDate.parse(finalDate, formatters);
                final long days = ChronoUnit.DAYS.between(firstDate, secondDate);
                Log.e( tag: "Days between: ", String.valueOf(days));

                if (Integer.parseInt(String.valueOf(days)) <= 7){
                    item_name.add(cursor.getString( columnIndex: 0));
                    item_qty.add(cursor.getInt( columnIndex: 1));
                    item_exp.add(finalDate);
                    item_category.add(cursor.getString( columnIndex: 3));
                    item_storage.add(cursor.getString( columnIndex: 4));
```

Figure 24: Expiry insert data to Array List

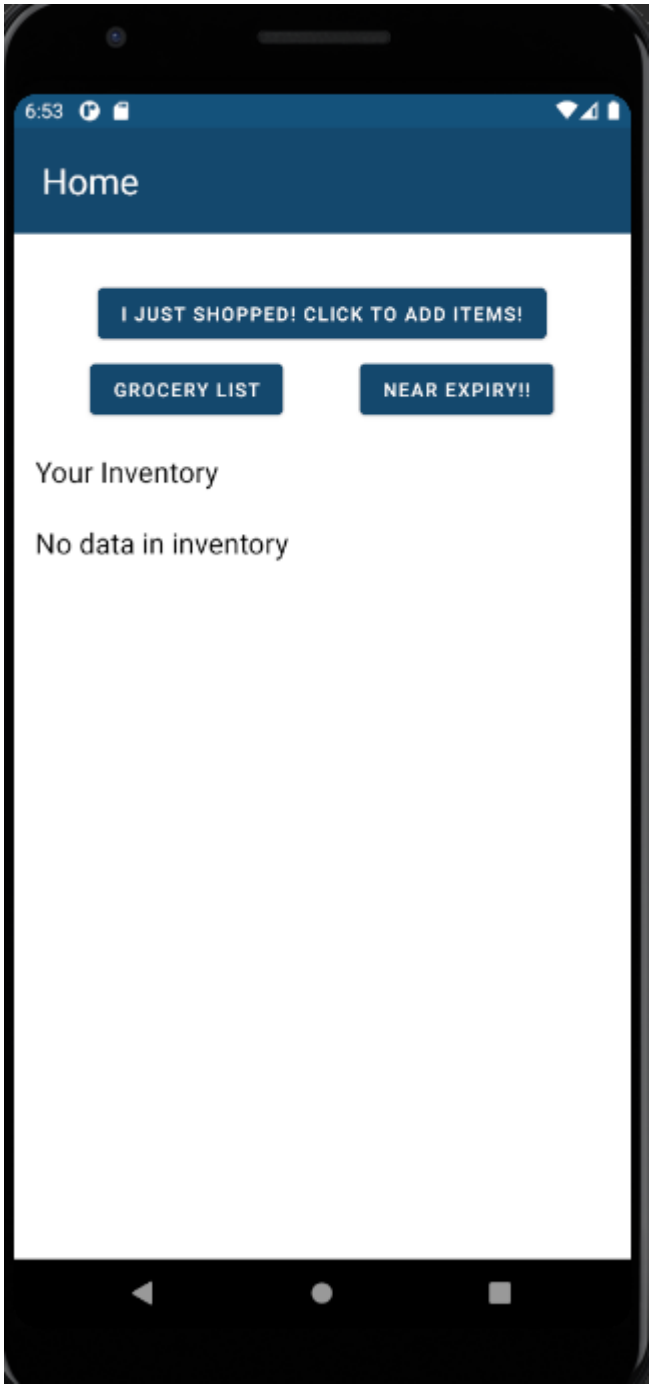
```
void storeDataInArrays() throws ParseException {
    Cursor cursor = db.readGroceryList();
    if(cursor.getCount() == 0){
        Toast.makeText( context: this,  text: "No data", Toast.LENGTH_SHORT).show();
    }else{
        Log.e( tag: "Listss",  msg: "have data");
        Log.e( tag: "Listss", cursor.toString());
        while (cursor.moveToNext()){
            if (cursor.getInt( columnIndex: 1) == 0){
                item_category.add(cursor.getString( columnIndex: 0));
                item_qty.add(cursor.getInt( columnIndex: 1));
            }
        }
    }
    Log.e( tag: "Listss", item_category.toString());
    Log.e( tag: "Listss", item_qty.toString());
}
```

Figure 25: Grocery list insert data to Array List

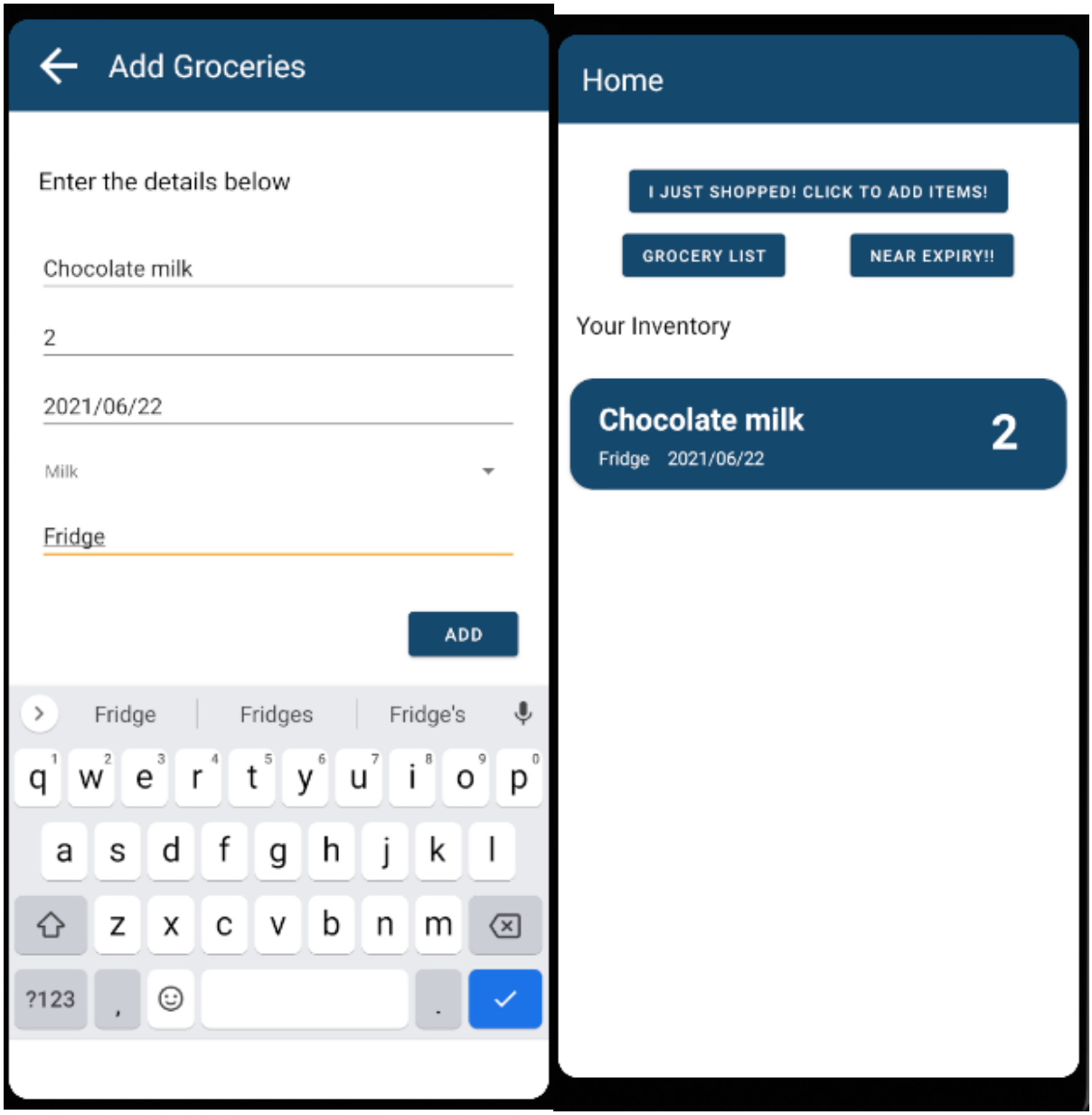
These two activities show the recycler view that helps users know which items will expire soon and which groceries are out of stock. The code is similar to the binding of recycler view in HomeActivity.java. The only difference is the code to store the data to the Array List. In expiry, a new filter is added where the expiry date is compared with the current date and the one with 7 or less days will be displayed. As for grocery activity, I made a new adapter as I wanted to use a different format for each data. I only want the category to show, instead of the whole information of the item. For the filter, I made it so that only categories with 0 quantity will be inserted into the array list and displayed in the recycler view.

D. Proof of working product

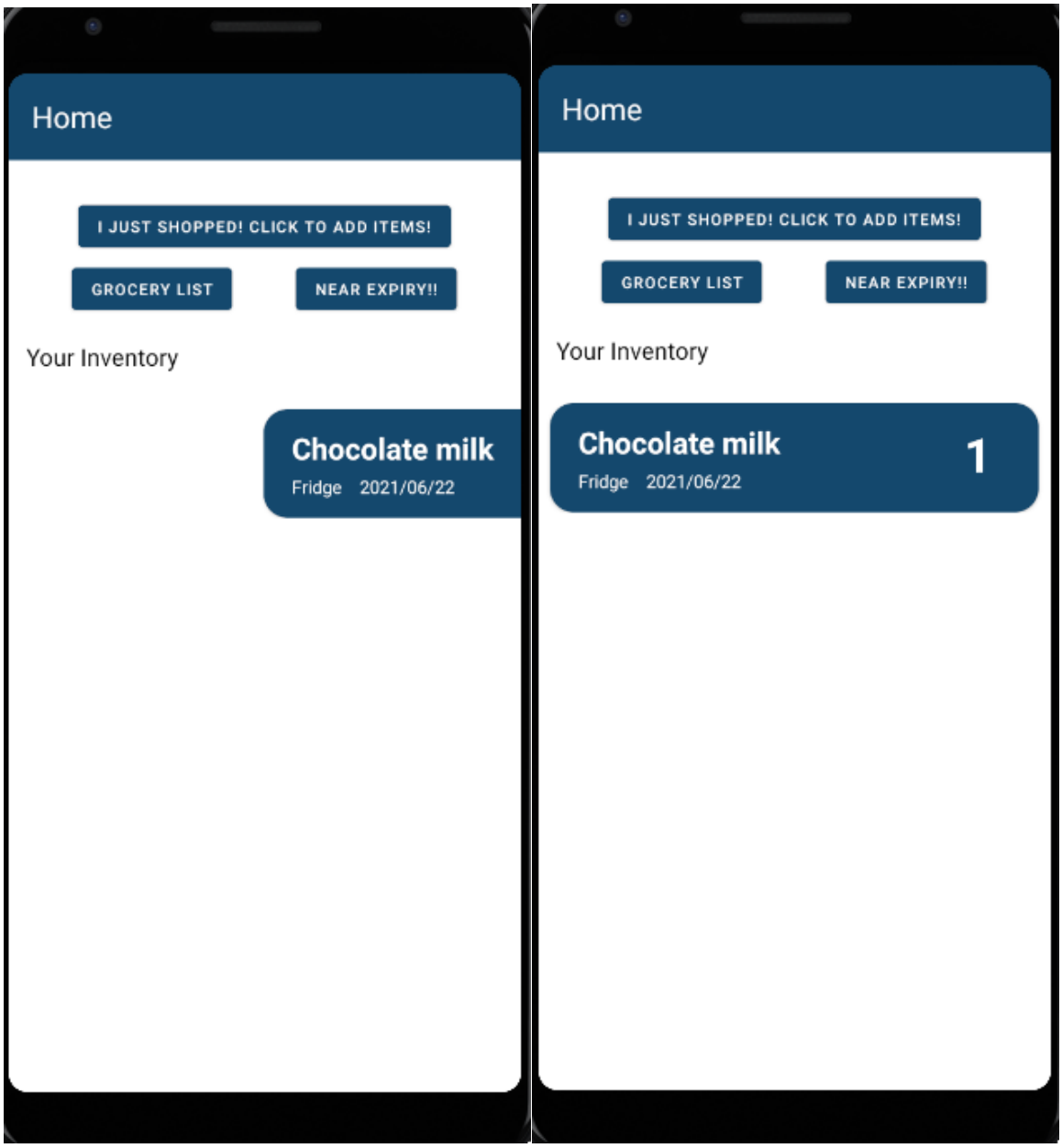
Home page when first launch



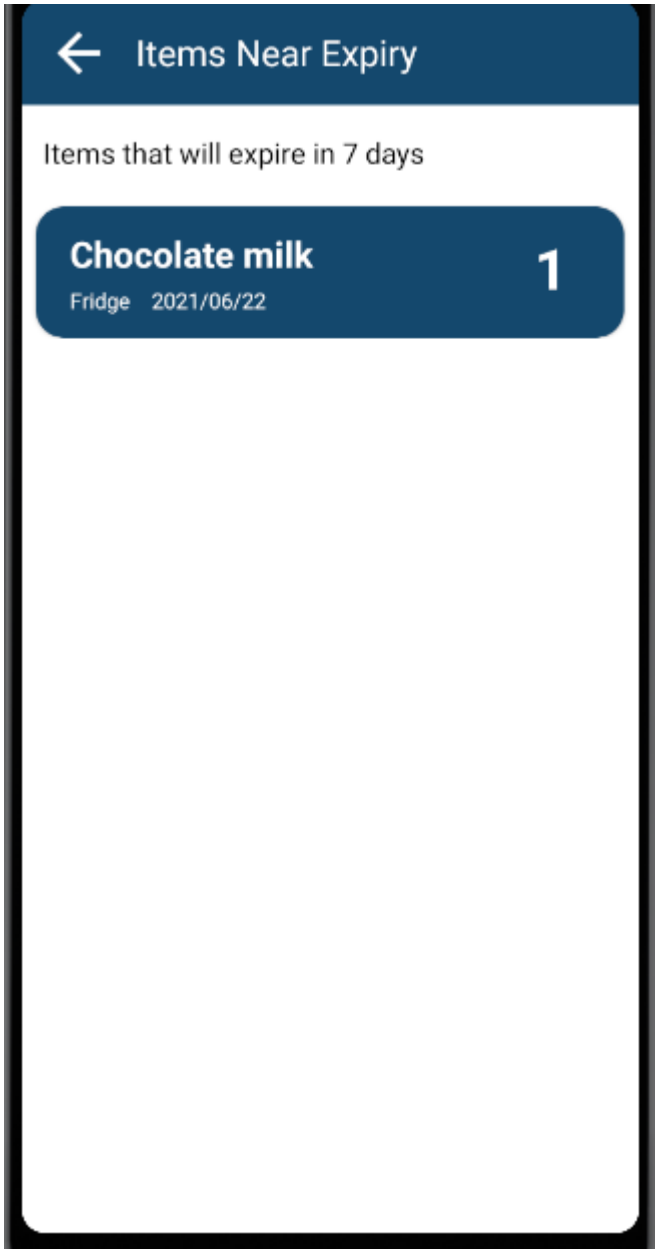
Inserting data



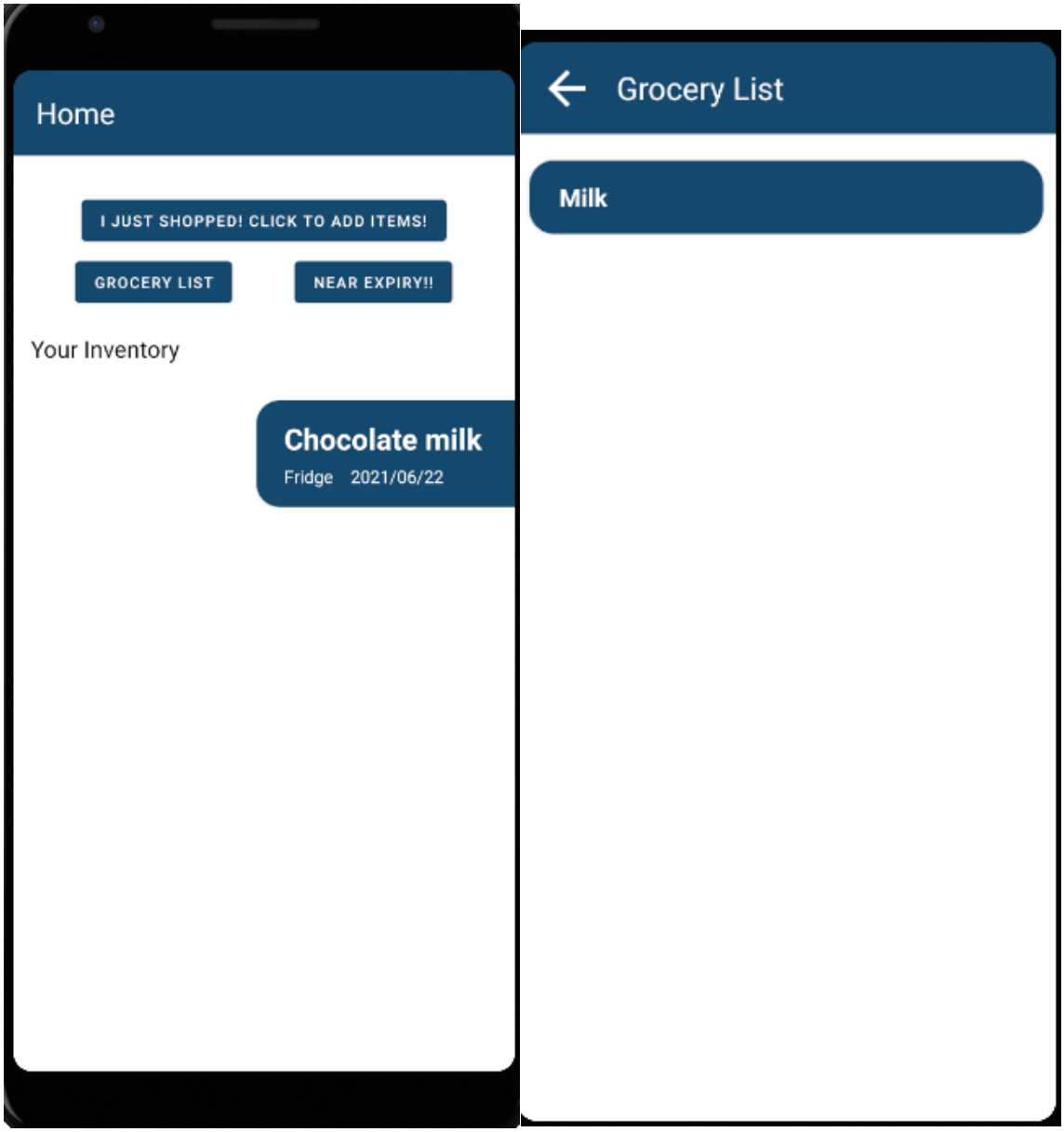
Updating data



Expiry


Items Near Expiry
Items that will expire in 7 days
Chocolate milk
Fridge 2021/06/22
1

Grocery list



When restock