CRITERION C

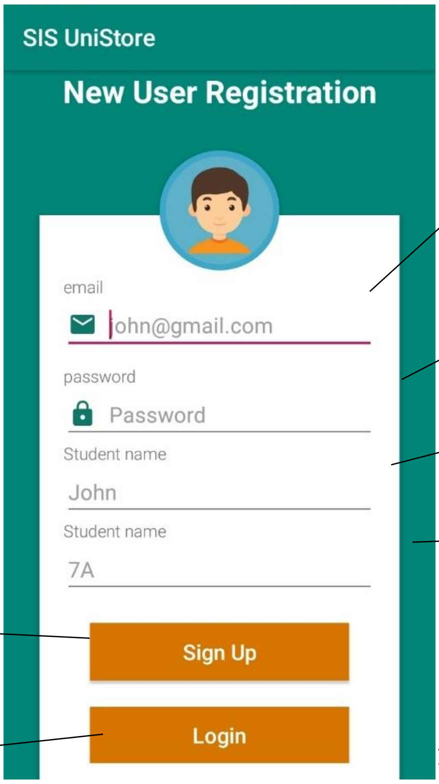
List of complexities achieved:

Sr.No.	Complexity	Purpose	Evidence (Criterion C figures)
1	Connecting to online database	Connect to Firebase Database for registration and login	Fig. 2A, 2B, 7, 8
2	Verifying the email id entered	To check if entered email id exists.	Fig. 4A,B
3	Forgot Password Mail	Send a forgot password link to reset the password	Fig. 9, Fig. 10A,B
3	Branching Statements	If-else statements to open a page/ display information according to the radio buttons selected	Fig. 12, 31, 33, 35, 36, 41, 43, 44
4	Linking activities in java	Connecting multiple activities through Intent methods and passing data between them	Fig. 6, 8, 12
5	Card View and Recycler View	To display each uniform item separately and link to its corresponding information	Fig. 14, 15, 19
6	Connect to offline database	Storing the selected/ entered details by the user	Fig. 21 – Fig. 33
7	Displaying the details stored in the offline database	The user can view the cart to know the uniform items selected by it.	Fig. 26 -27
8	Displaying a pop-up notification	A pop-up notification of inbox type after confirming the order.	Fig. 44, 45

Text in Blue – Reference to figure in **Criterion B**

Text in red – Reference to figures in **Criterion**

1. Registration Page



SIS UniStore

New User Registration

email

john@gmail.com

password

Password

Student name

John

Student name

7A

Sign Up

Login

[1]

Edit Textbox: Enter Email id to register with.

Edit Textbox: Enter Password

Edit Textbox: Enter name of student

Edit Textbox: Enter Grade

Sign up the entered Email id. Send a verification mail to the above email id. Linked through Firebase Database.

Button to go to the Login Page for Registered users.

Fig. 1 Fig. 1

Creates new user with email and password in the online database. This links the Registration Page to the Firebase Online Database

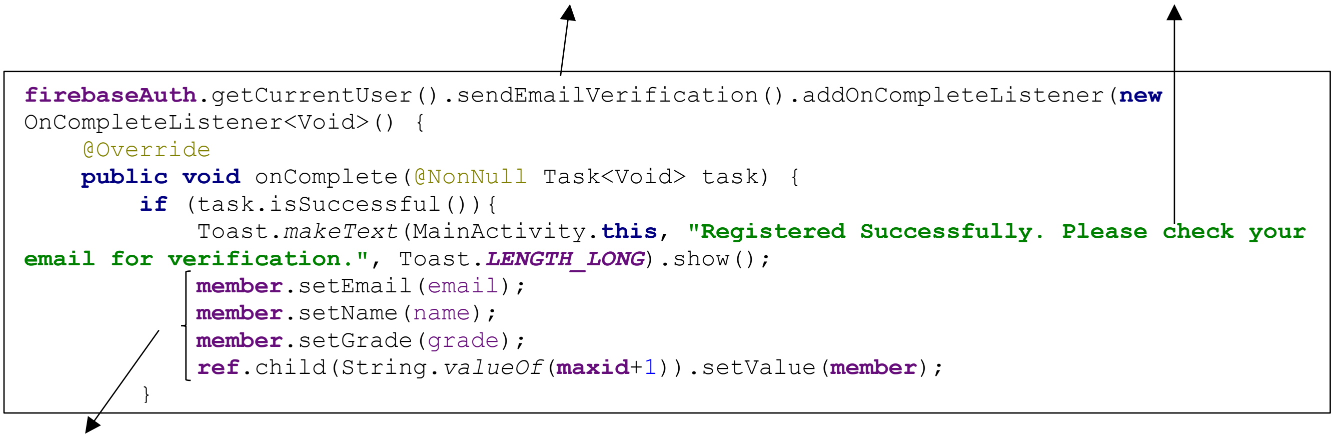
```
firebaseAuth.createUserWithEmailAndPassword(email,
pwd).addOnCompleteListener(MainActivity.this, new OnCompleteListener<AuthResult>()
```

Fig. 2A

[1] UI - https://www.simplifiedcoding.net/firebase-authentication-tutorial/#Firebase-Authentication-Tutorial-8211-Source-Code

Email Verification sent to the registered email id to check if the email id exists.

Message displayed to check the mail inbox and verify the email.

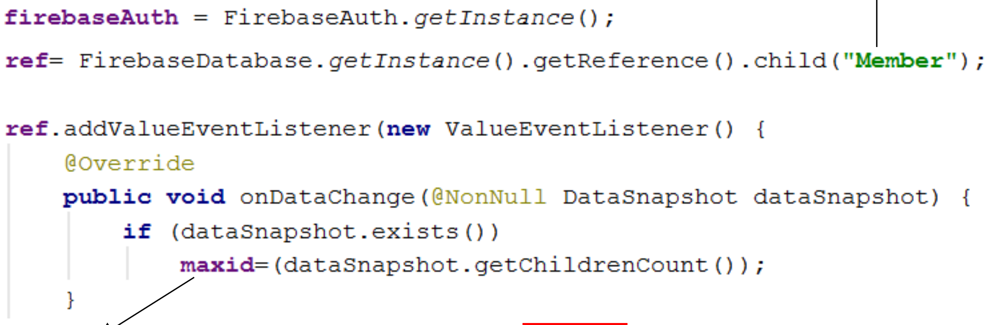


```
firebaseAuth.getCurrentUser().sendEmailVerification().addOnCompleteListener(new
OnCompleteListener<Void>() {
    @Override
    public void onComplete(@NonNull Task<Void> task) {
        if (task.isSuccessful()){
            Toast.makeText(MainActivity.this, "Registered Successfully. Please check your
email for verification.", Toast.LENGTH_LONG).show();
            member.setEmail(email);
            member.setName(name);
            member.setGrade(grade);
            ref.child(String.valueOf(maxid+1)).setValue(member);
        }
```

Adds the member fields of the ‘Member Table’ in Firebase Database.

Fig. 2B

Table ‘Member’ constructed in firebase database

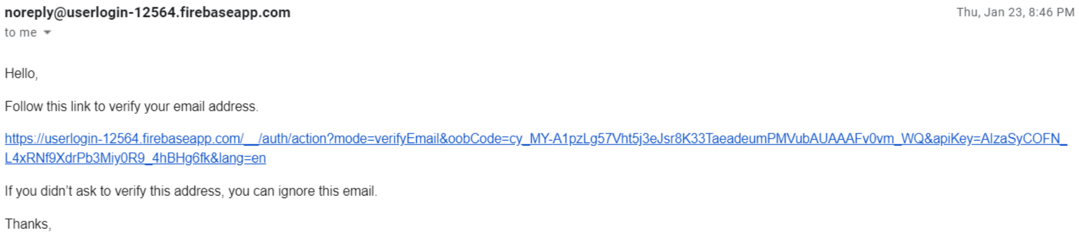


```
firebaseAuth = FirebaseAuth.getInstance();
ref= FirebaseDatabase.getInstance().getReference().child("Member");

ref.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        if (dataSnapshot.exists())
            maxid=(dataSnapshot.getChildrenCount());
    }
```

Increments to add the different members that get registered.

Fig. 3

Evidence of Email Verification



noreply@userlogin-12564.firebaseapp.com
to me

Thu, Jan 23, 8:46 PM

Hello,

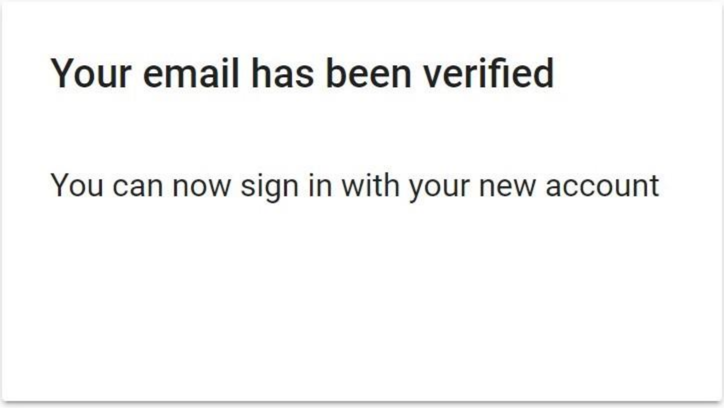
Follow this link to verify your email address.

https://userlogin-12564.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=cy_MY-A1pzLg57Vht5j3eJsr8K33TaeadeumPMVubAUAAAFv0vm_WQ&apiKey=AIzaSyCOFN_L4xRNf9XdrPb3Miy0R9_4hBHg6fk&lang=en

If you didn’t ask to verify this address, you can ignore this email.

Thanks,

Fig. 4A

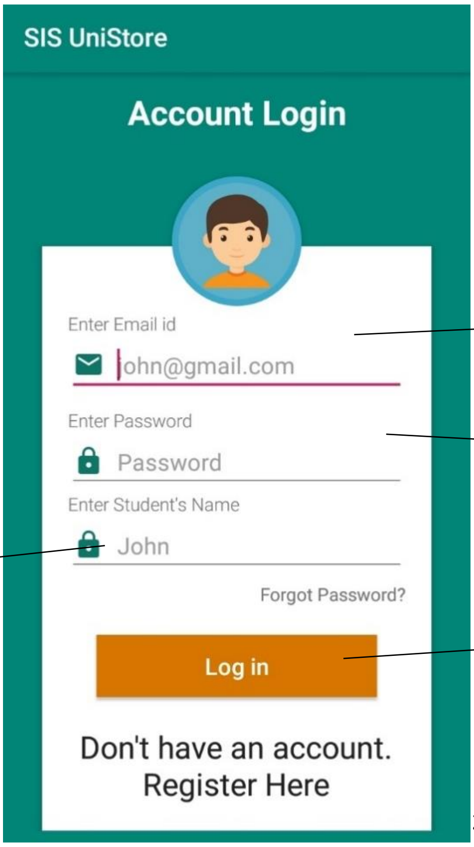


Your email has been verified

You can now sign in with your new account

Fig. 4B

2. Login Page



[2]

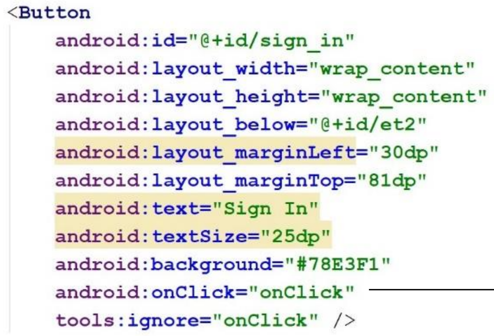
Edit Textbox: Enter Email id

Edit Textbox: Enter Password

Button Widget: When clicked goes to the next page. The password is checked and displays error toast messages. Check for verification of email id is also made.

Clickable Text to go to the Register Page for unregistered users. Verified through Firebase Database.

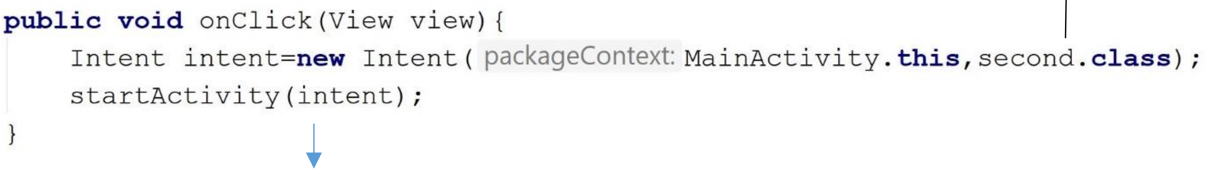
Fig. 5 Fig. 2



```
<Button
    android:id="@+id/sign_in"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/et2"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="81dp"
    android:text="Sign In"
    android:textSize="25dp"
    android:background="#78E3F1"
    android:onClick="onClick"
    tools:ignore="onClick" />
```

onClick command to go to the next page; defines the click method in the java code

Target class: this activity is launched when the button is clicked.



```
public void onClick(View view){
    Intent intent=new Intent( packageContext: MainActivity.this,second.class);
    startActivity(intent);
}
```

[2] UI - https://www.simplifiedcoding.net/firebase-authentication-tutorial/#Firebase-Authentication-Tutorial-8211-Source-Code

Starts the target activity that is to be launched

Fig. 6

Sign in method linked to the firebase database.

```
firebaseAuth.signInWithEmailAndPassword(email,pwd).addOnCompleteListener(login.this, new
OnCompleteListener<AuthResult>()
```

Fig. 7

Checks if the email is verified by the user by the email sent to the user like figure ().

```
if (firebaseAuth.getCurrentUser().isEmailVerified()) {
    Intent intToHome = new Intent(login.this, second.class);
    intToHome.putExtra("studentName", name);
    startActivity(intToHome);
}
else{
    Toast.makeText(login.this, "Please verify your email address",
Toast.LENGTH_LONG).show();
```

Sends toast message to verify the email.

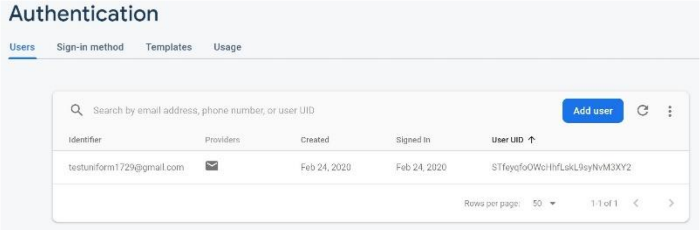
Fig. 8A

The View[3] is the building block for user interface components. It occupies a rectangular area on the screen and is responsible for drawing and event handling. It is a base class for all widgets as well. Here also it occupies space for the specific button.

- Intents are messages which allow components of the application to request functionality from other components of that application. There are two types of Intents: Explicit and Implicit. Here, Explicit intent is used to start the target activity.

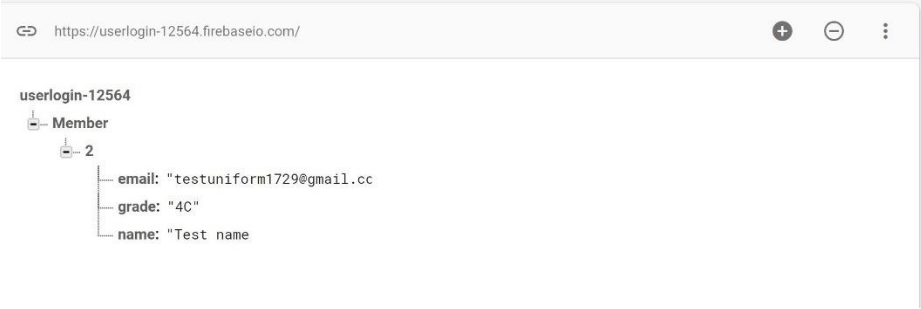
[3] https://developer.android.com/reference/android/view/View.html

Evidence of Online Firebase Realtime Database



Identifier	Providers	Created	Signed In	User UID ↑
testuniform1729@gmail.com	✉	Feb 24, 2020	Feb 24, 2020	STfeyqfoOWcHhfLskL9syNvM3XY2

Fig. 8B

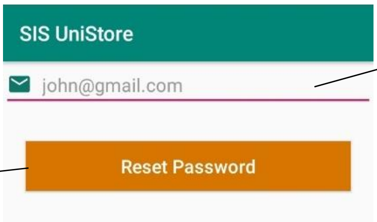


```
userlogin-12564
 ⊟ Member
    ⊟ 2
       email: "testuniform1729@gmail.cc
       grade: "4C"
       name: "Test name
```

Fig. 8C

3. Forgot Password

Links to the Firebase Database and sends a password reset mail to the entered email id.

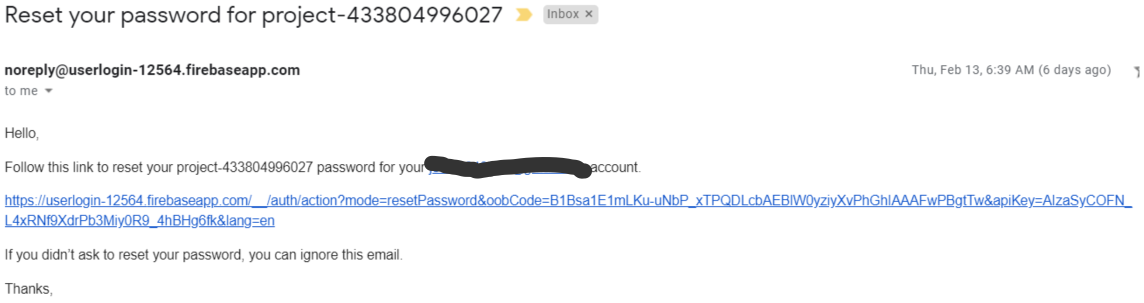


Edit TextBox:

Enter email id whose password needs to be reset.

Fig. 9

Evidence of Reset Password mail



Reset your password for project-433804996027 Inbox ×

noreply@userlogin-12564.firebaseapp.com
to me

Thu, Feb 13, 6:39 AM (6 days ago)

Hello,

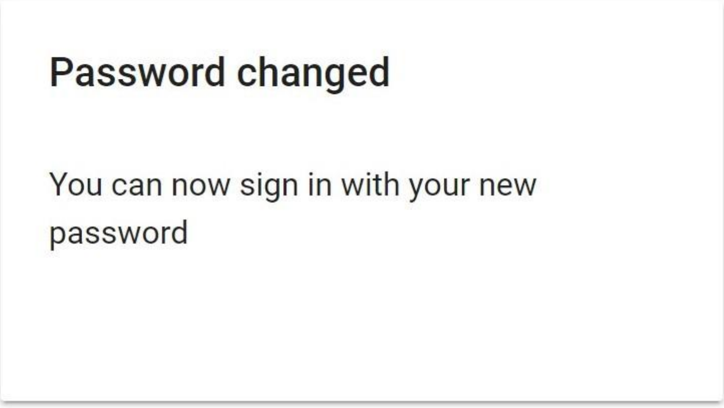
Follow this link to reset your project-433804996027 password for your [redacted] account.

https://userlogin-12564.firebaseapp.com/__/auth/action?mode=resetPassword&oobCode=B1Bsa1E1mLKu-uNbP_xTPQDLcbAEBlW0yziyXvPhGhlAAAFwPBgtTw&apiKey=AIzaSyCOFN_L4xRNf9XdrPb3Miy0R9_4hBHg6fk&lang=en

If you didn't ask to reset your password, you can ignore this email.

Thanks,

Fig. 10A

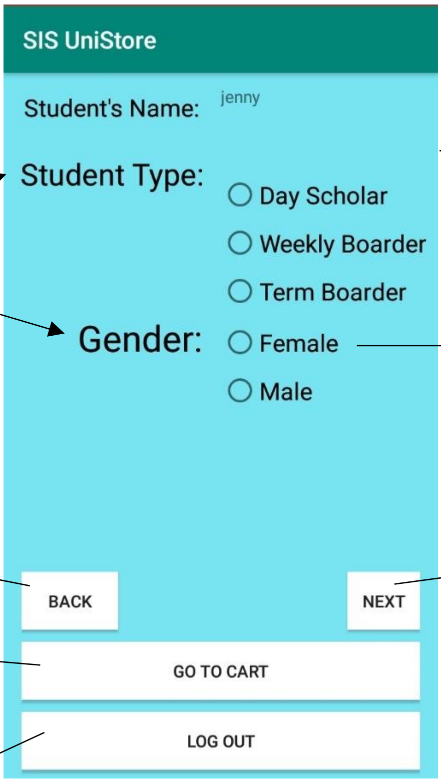


Password changed

You can now sign in with your new password

Fig. 10B

4. Second Page



Student's Name: jenny

Student Type:
○ Day Scholar
○ Weekly Boarder
○ Term Boarder

Gender:
○ Female
○ Male

Fig. 11 Fig. 3

2 Radio Groups

Radio Buttons

Returns to the previous page

Button to go to the cart

Logs out the current user. Linking to the Firebase Database.

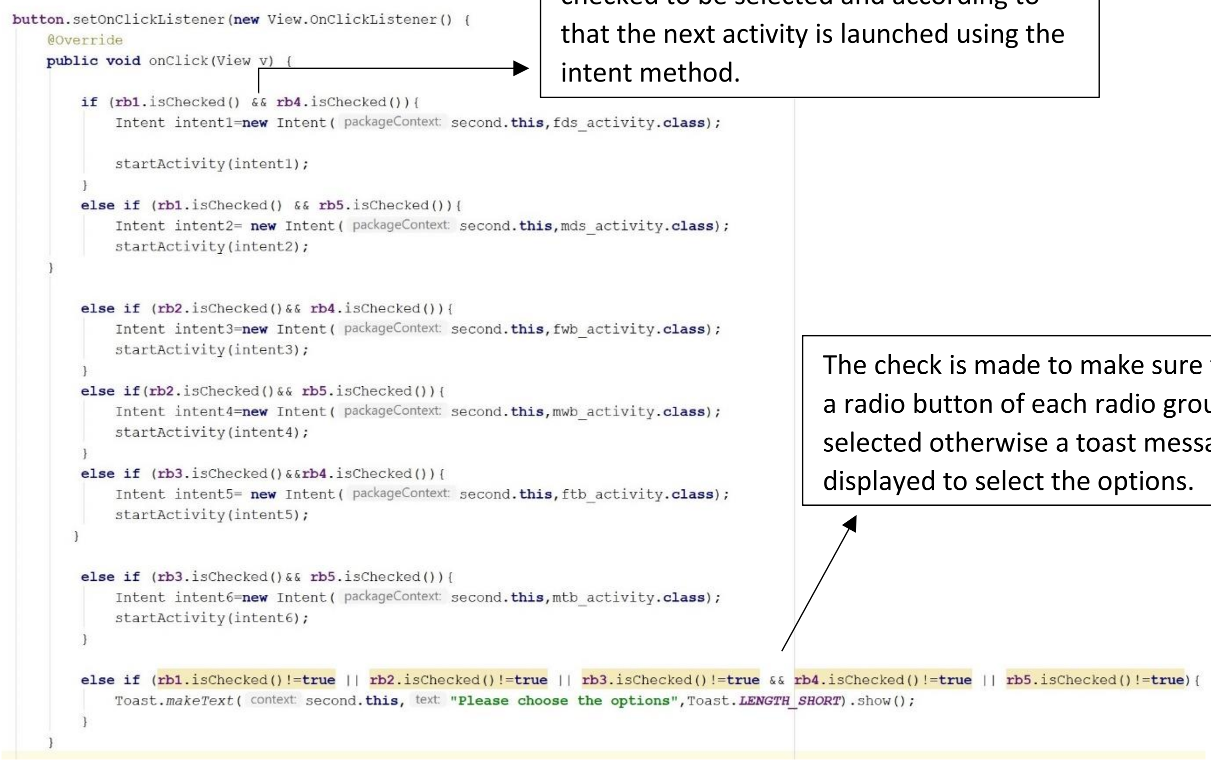
Button: Launches the next activity by using the else-if method. The radio button chosen in the 2 radio groups defines the next activity to be launched. Sends a toast message if a radio button each radio group isn’t selected. Intent method used to launch the next activity.

A similar setOnClickListener is created for the 'Back' Button and 'Go to Cart' Button

onClick(View v):

It is an abstract method. The code written inside this method is implemented when the button is clicked.

The radio button of first radio group and radio button of second radio group is checked to be selected and according to that the next activity is launched using the intent method.



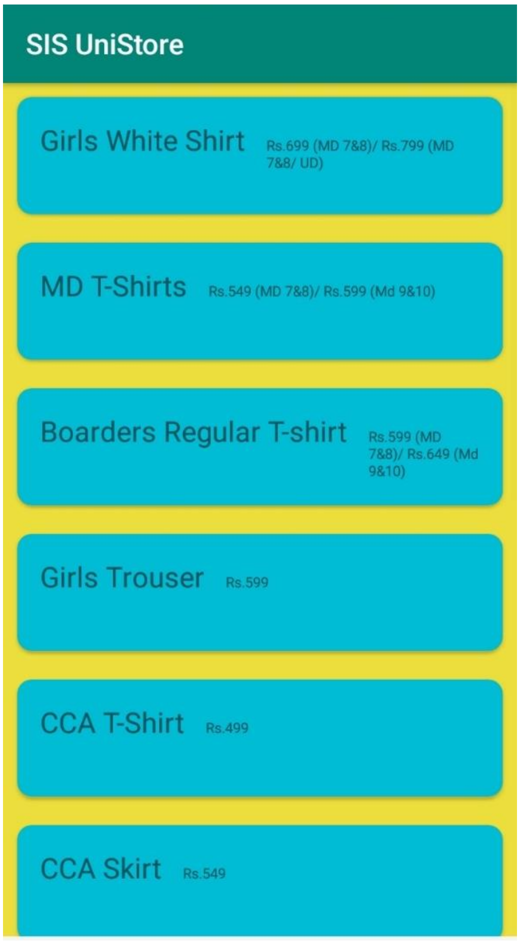
```java
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        if (rb1.isChecked() && rb4.isChecked()){
            Intent intent1=new Intent( packageContext: second.this,fds_activity.class);

            startActivity(intent1);
        }
        else if (rb1.isChecked() && rb5.isChecked()){
            Intent intent2= new Intent( packageContext: second.this,mds_activity.class);
            startActivity(intent2);
    }

        else if (rb2.isChecked()&& rb4.isChecked()){
            Intent intent3=new Intent( packageContext: second.this,fwb_activity.class);
            startActivity(intent3);
        }
        else if(rb2.isChecked()&& rb5.isChecked()){
            Intent intent4=new Intent( packageContext: second.this,mwb_activity.class);
            startActivity(intent4);
        }
        else if (rb3.isChecked()&&rb4.isChecked()){
            Intent intent5= new Intent( packageContext: second.this,ftb_activity.class);
            startActivity(intent5);
       }

        else if (rb3.isChecked()&& rb5.isChecked()){
            Intent intent6=new Intent( packageContext: second.this,mtb_activity.class);
            startActivity(intent6);
        }

        else if (rb1.isChecked()!=true || rb2.isChecked()!=true || rb3.isChecked()!=true && rb4.isChecked()!=true || rb5.isChecked()!=true){
            Toast.makeText( context: second.this, text: "Please choose the options",Toast.LENGTH_SHORT).show();
        }
    }
```

The check is made to make sure that a radio button of each radio group is selected otherwise a toast message is displayed to select the options.

Fig. 12

5. List Items Page

- This activity is launched according to the options chosen.
- This activity consists the list of items that the user can select and buy.
- The items in the list are clickable.



SIS UniStore

Girls White Shirt Rs.699 (MD 7&8)/ Rs.799 (MD 7&8/ UD)

MD T-Shirts Rs.549 (MD 7&8)/ Rs.599 (Md 9&10)

Boarders Regular T-shirt Rs.599 (MD 7&8)/ Rs.649 (Md 9&10)

Girls Trouser Rs.599

CCA T-Shirt Rs.499

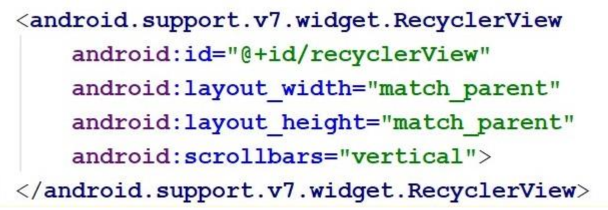
CCA Skirt Rs.549

Fig 13 Fig. 4

To create this activity two widgets were used:

It is an advanced and flexible version of List View. The advantage of using recycler view is that it renders large data sets that can be scrolled efficiently saving memory space.

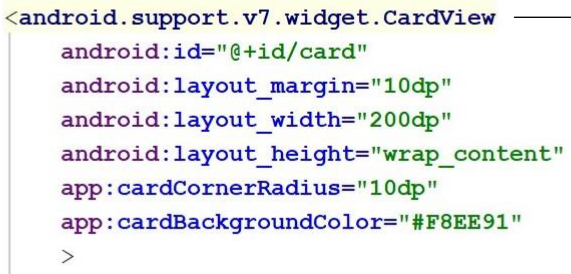
1) Recycler View-



```
<android.support.v7.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbars="vertical">
</android.support.v7.widget.RecyclerView>
```

Fig. 14

2) Card View



```
<android.support.v7.widget.CardView
    android:id="@+id/card"
    android:layout_margin="10dp"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    app:cardCornerRadius="10dp"
    app:cardBackgroundColor="#F8EE91"
    >
```

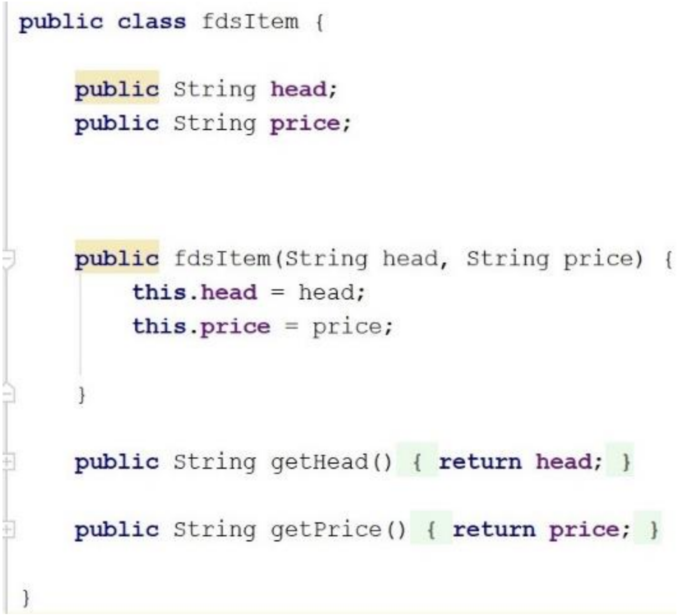
The Card View gives a uniform layout to the card-based UI's. It allows the use of different views like text view, image view, etc. in a uniform way.

Fig. 15

For each activity, three classes were created:

I. fds_item
II. fds_adapter
III. fds_activity

fds item class



```
public class fdsItem {

    public String head;
    public String price;



    public fdsItem(String head, String price) {
        this.head = head;
        this.price = price;

    }

    public String getHead() { return head; }

    public String getPrice() { return price; }

}
```

Fig. 16

- Accessor method used to access the string objects: head and price created in the fds_list.xml. They are accessed to make a list in the fds_activity class. There are many object so to add a new object these two strings will be used along with the parameterized constructor.

Java Class: fds_adapter

Recycler View Adapter bind the data of to the view that is to be displayed within the Recycler View.

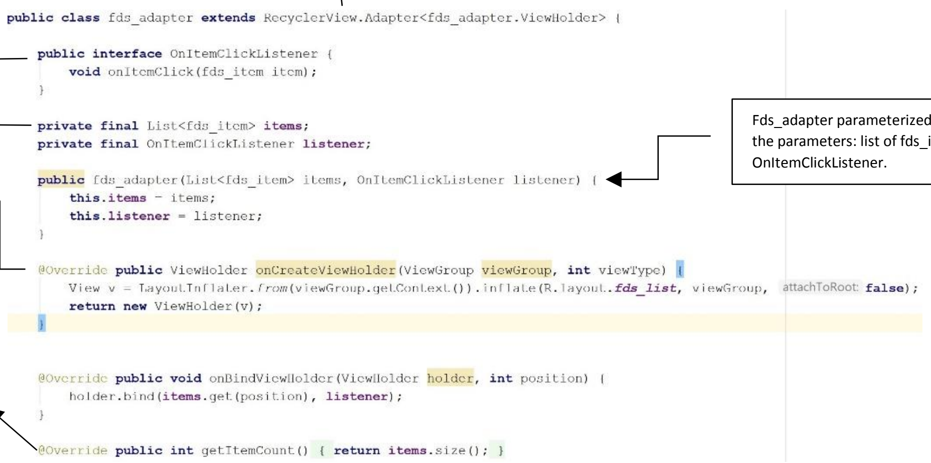
It is invoked when an item in the adapter view is clicked. The parameter calls the fds_item class while creating an object of it which makes that item clickable.

A list called 'items' of fds_item constructor is created.

onCreateViewHolder method has return type ViewHolder.

Fds_adapter parameterized constructor created with the parameters: list of fds_item and listener of the OnItemClickListener.

getItemCount() method has return type 'Integer.' It returns the total number of items in the list i.e. the size of items is returned.



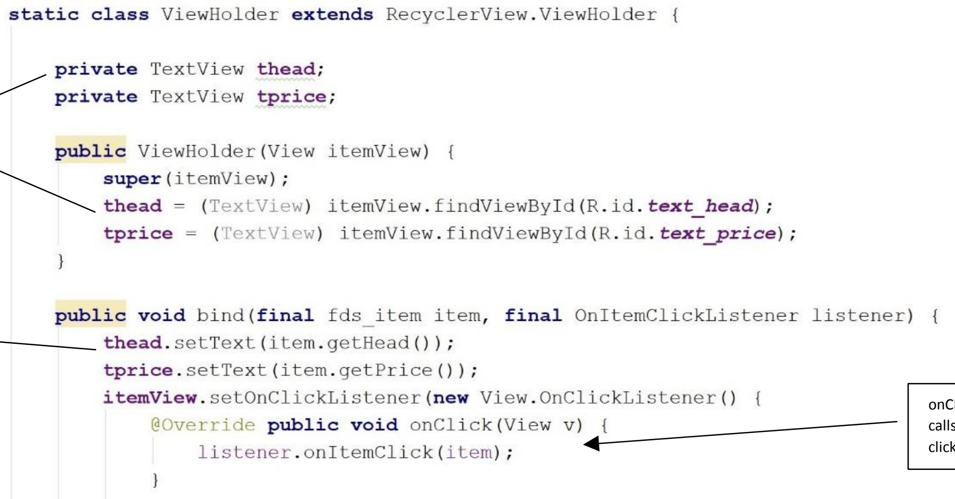
```
public class fds_adapter extends RecyclerView.Adapter<fds_adapter.ViewHolder> {

    public interface OnItemClickListener {
        void onItemClick(fds_item item);
    }

    private final List<fds_item> items;
    private final OnItemClickListener listener;

    public fds_adapter(List<fds_item> items, OnItemClickListener listener) {
        this.items = items;
        this.listener = listener;
    }

    @Override public ViewHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.fds_list, viewGroup,  attachToRoot: false);
        return new ViewHolder(v);
    }


    @Override public void onBindViewHolder(ViewHolder holder, int position) {
        holder.bind(items.get(position), listener);
    }

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

Fig. 17

TextView of the xml file defined. It is called by the itemView to define where the text is to be placed in the xml file.

setText method used to define the text to be displayed. The parameter passed is accessor method called from the fds_item class.

onCLickListener set on itemView to make the recycler view clickable. Listener calls the onItemCLick method and passes item to make the item of the list clickable.



```
static class ViewHolder extends RecyclerView.ViewHolder {

    private TextView thead;
    private TextView tprice;

    public ViewHolder(View itemView) {
        super(itemView);
        thead = (TextView) itemView.findViewById(R.id.text_head);
        tprice = (TextView) itemView.findViewById(R.id.text_price);
    }

    public void bind(final fds_item item, final OnItemClickListener listener) {
        thead.setText(item.getHead());
        tprice.setText(item.getPrice());
        itemView.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View v) {
                listener.onItemClick(item);
            }
```

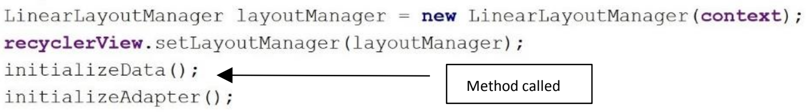
Fig. 18 (continued part of the fds_adapter class)

Java Class: fds_activity

It is used to create a vertical Linear Layout Manager. In this we need to set only one parameter i.e. used to set the context the current Activity.

The layout manager of recycler view is set by calling 'layoutManager'

Method called



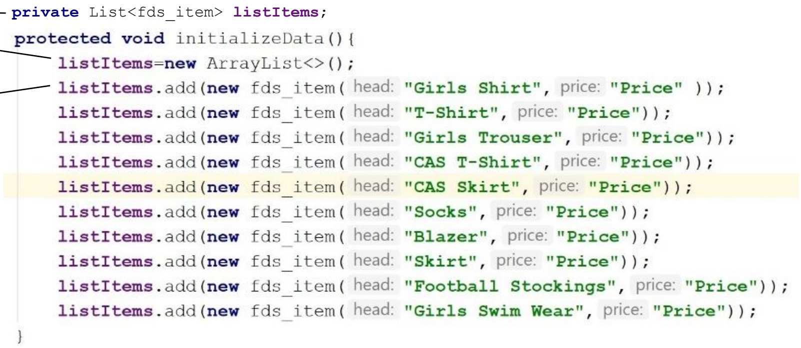
```
LinearLayoutManager layoutManager = new LinearLayoutManager(context);
recyclerView.setLayoutManager(layoutManager);
initializeData();
initializeAdapter();
```

Fig. 19

Empty Array List called 'listItems' of 'fds_item' class constructed.

New items added to the list by invoking the parameterized constructor of fds_item class.

For each activity the list varies. For instance, the mds (male-day scholar) contains Boys Shirt and the Boarding i.e. fwb (female-weekly boarder) would have boarding uniform options also available. The list varies according to the options selected in second (Refer to Fig. 8)



```
private List<fds_item> listItems;
protected void initializeData(){
    listItems=new ArrayList<>();
    listItems.add(new fds_item( head: "Girls Shirt", price: "Price" ));
    listItems.add(new fds_item( head: "T-Shirt", price: "Price"));
    listItems.add(new fds_item( head: "Girls Trouser", price: "Price"));
    listItems.add(new fds_item( head: "CAS T-Shirt", price: "Price"));
    listItems.add(new fds_item( head: "CAS Skirt", price: "Price"));
    listItems.add(new fds_item( head: "Socks", price: "Price"));
    listItems.add(new fds_item( head: "Blazer", price: "Price"));
    listItems.add(new fds_item( head: "Skirt", price: "Price"));
    listItems.add(new fds_item( head: "Football Stockings", price: "Price"));
    listItems.add(new fds_item( head: "Girls Swim Wear", price: "Price"));
}
```

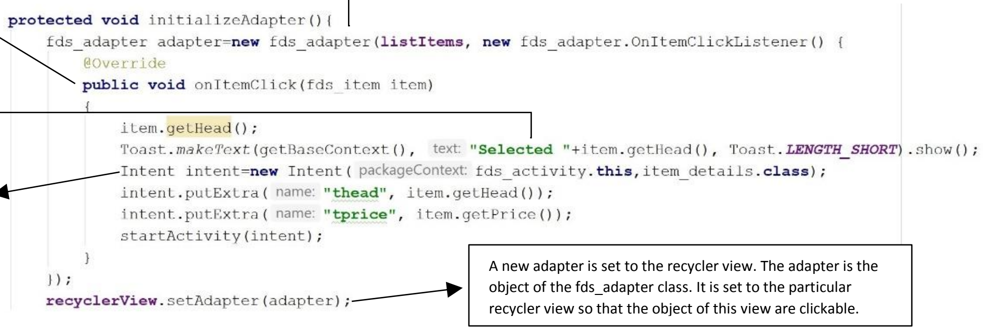
Fig. 20

An object of 'fds_adapter' class created which passes two parameters: first, the array list and an onClickListener for the item, which makes it clickable.

Method to define the action on clicking the item passed in the parameter.

A toast message displayed with the name of the item selected as the message.

Moves to another activity/page when the item is clicked using the intent method. It also sets the item name and price using putExtra method to retrieve it in the next page.



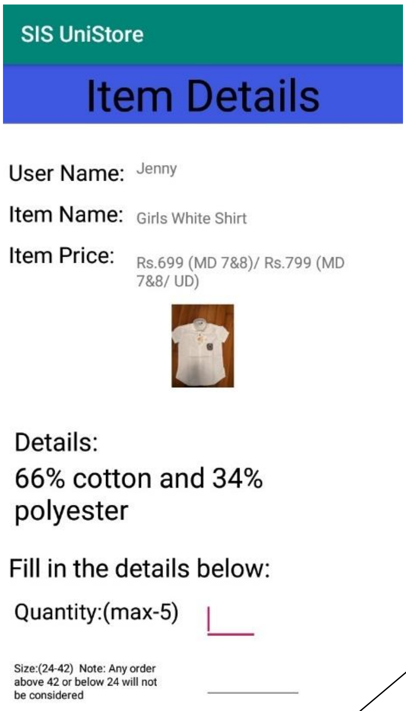
```
protected void initializeAdapter(){
    fds_adapter adapter=new fds_adapter(listItems, new fds_adapter.OnItemClickListener() {
        @Override
        public void onItemClick(fds_item item)
        {
            item.getHead();
            Toast.makeText(getBaseContext(), text: "Selected "+item.getHead(), Toast.LENGTH_SHORT).show();
            Intent intent=new Intent( packageContext: fds_activity.this,item_details.class);
            intent.putExtra( name: "thead", item.getHead());
            intent.putExtra( name: "tprice", item.getPrice());
            startActivity(intent);
        }
    });
    recyclerView.setAdapter(adapter);
```

A new adapter is set to the recycler view. The adapter is the object of the fds_adapter class. It is set to the particular recycler view so that the object of this view are clickable.

Fig.21

6. Item Details Page

- This is the page that opens up when the item in the list of card view is clicked.
- It displays all the details of the item and is again interactive with the user since the user has to mention the quantity and size of the uniform they have selected.
- The item details java class is related to the DatabseHelper class.
- The Database Helper Class creates SQL Database to store, retrieve and delete data i.e. the uniform products.



SIS UniStore

Item Details

User Name: Jenny

Item Name: Girls White Shirt

Item Price: Rs.699 (MD 7&8)/ Rs.799 (MD 7&8/ UD)

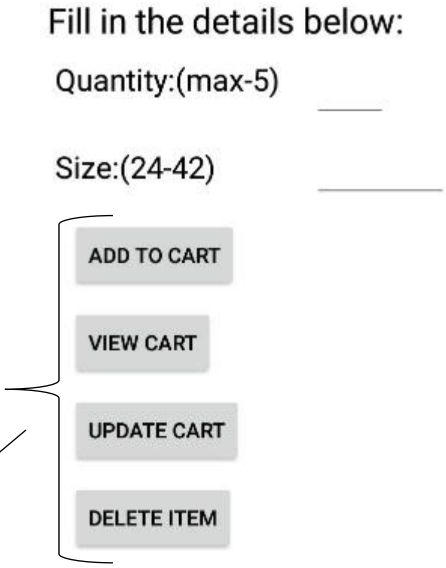
Details:
66% cotton and 34% polyester

Fill in the details below:

Quantity:(max-5)

Size:(24-42) Note: Any order above 42 or below 24 will not be considered

Fig. 22 Fig. 5



Fill in the details below:

Quantity:(max-5)

Size:(24-42)

Fig. 23(cont. of 22) Fig. 5

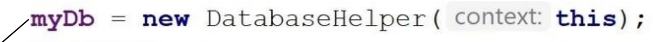
The functionality of each button is displayed on the pages(). The left column of the tables shown on these pages() depict the front end functionality and the right column depict the way in which the data is stored in the offline Database MySQL.

Java Class: item_details

Java Class: DatabaseHelper

SQLiteOpenHelper is a class that manages the creation of a local database.

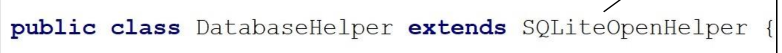
Name of the Buttons



```
myDb = new DatabaseHelper( context: this);
```

A new Database called ‘myDb’ constructed by calling

Fig. 24



```
public class DatabaseHelper extends SQLiteOpenHelper {
```

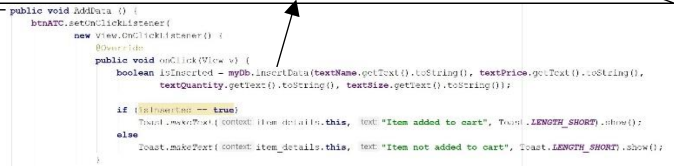
Fig. 25A

An object of SQLiteDatabase constructed that stores the getWritableDatabase method .This method creates/opens a database that would be used to read and write

The method has boolean return type. It is created to insert data in the database.

Boolean object created that calls the insertData method in DatabaseHelper class.

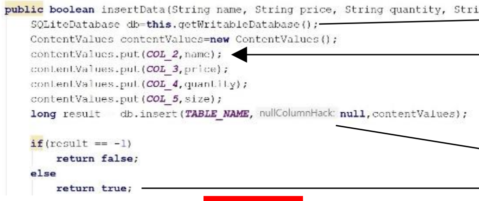
Add to Cart



```
public void AddData () {
    btnATC.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    boolean isInserted = myDb.insertData(textName.getText().toString(), textPrice.getText().toString(),
                            textQuantity.getText().toString(), textSize.getText().toString());

                    if (isInserted == true)
                        Toast.makeText( context: item_details.this,  text: "Item added to cart", Toast.LENGTH_SHORT).show();
                    else
                        Toast.makeText( context: item_details.this,  text: "Item not added to cart", Toast.LENGTH_SHORT).show();
                }
```

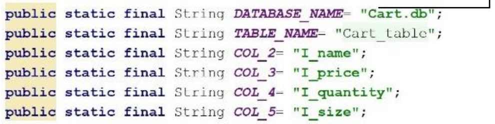
Fig. 26

To add an item to the cart when the user clicks the add item button (Refer Fig. 20). The new item has to be added once the button is clicked. Hence, the inserted action will be written under the OnClickListener method.



```
public boolean insertData(String name, String price, String quantity, String size){
    SQLiteDatabase db=this.getWritableDatabase();
    ContentValues contentValues=new ContentValues();
    contentValues.put(COL_2,name);
    contentValues.put(COL_3,price);
    contentValues.put(COL_4,quantity);
    contentValues.put(COL_5,size);
    long result = db.insert(TABLE_NAME, nullColumnHack: null,contentValues);

    if(result == -1)
        return false;
    else
        return true;
```

Fig. 27



```
public static final String DATABASE_NAME= "Cart.db";
public static final String TABLE_NAME= "Cart_table";
public static final String COL_2= "I_name";
public static final String COL_3= "I_price";
public static final String COL_4= "I_quantity";
public static final String COL_5= "I_size";
```

Fig. 28

An object of Content Values class constructed. Content Values stores a set of data.

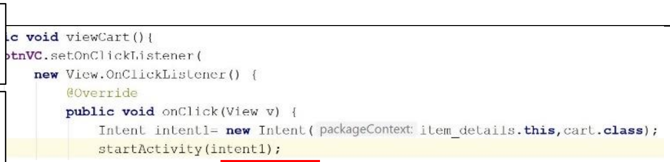
Data can be input in the content values using the ‘put’ method. It takes two parameters: first, the String object defined to store the value, i.e. ‘COL_2’ and second, the key/name of the string.

An object called ‘result’ with long data type invokes insert method to insert the new row in the database.

View Cart

OnClickListener constructed to present some action on clicking the button 'View Cart' (Refer Fig. 20)

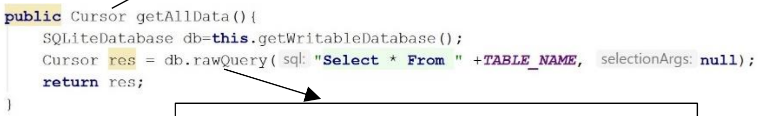
Intent method used to go to another activity when the button is clicked. In the target activity i.e. 'Cart' the class will call the database to retrieve data from it. Hence, the code to retrieve data from database is written in the 'cart' java class.



```
ic void viewCart(){
btnVC.setOnClickListener(
        new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent1= new Intent( packageContext: item_details.this,cart.class);
                startActivity(intent1);
```

Fig. 29

getAllData() method used to retrieve data from the database present.



```
public Cursor getAllData(){
    SQLiteDatabase db=this.getWritableDatabase();
    Cursor res = db.rawQuery( sql: "Select * From " +TABLE_NAME,  selectionArgs: null);
    return res;
}
```

'rawQuery' method runs a query and returns a cursor over the result.

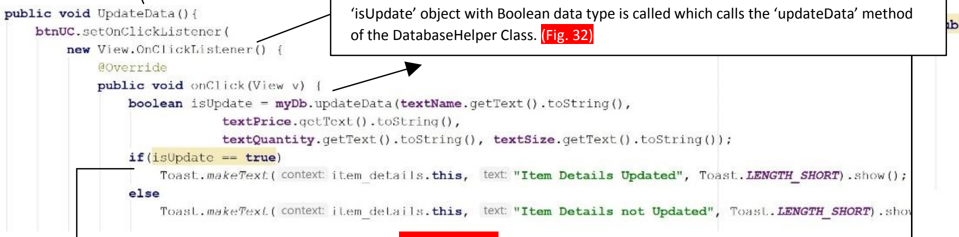
Fig. 30

Update cart

To update the data like quantity and size of a uniform selected in the cart.

The data is updated once the button is clicked. Hence, the update method is under the OnClickListener method of the 'Update Cart' Button

'isUpdate' object with Boolean data type is called which calls the 'updateData' method of the DatabaseHelper Class. (Fig. 32)

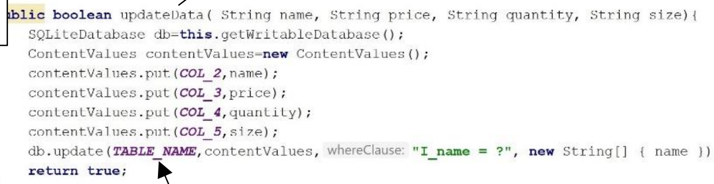


```
public void UpdateData(){
    btnUC.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    boolean isUpdate = myDb.updateData(textName.getText().toString(),
                            textPrice.getText().toString(),
                            textQuantity.getText().toString(), textSize.getText().toString());
                    if(isUpdate == true)
                        Toast.makeText( context: item_details.this,  text: "Item Details Updated", Toast.LENGTH_SHORT).show();
                    else
                        Toast.makeText( context: item_details.this,  text: "Item Details not Updated", Toast.LENGTH_SHORT).show
```

Fig. 31

When the result returned is true it indicates that the data in the cart has been updated which is shown through a toast message.

updateData method has return type, Boolean.



```
blic boolean updateData( String name, String price, String quantity, String size){
    SQLiteDatabase db=this.getWritableDatabase();
    ContentValues contentValues=new ContentValues();
    contentValues.put(COL_2,name);
    contentValues.put(COL_3,price);
    contentValues.put(COL_4,quantity);
    contentValues.put(COL_5,size);
    db.update(TABLE_NAME,contentValues, whereClause: "I_name = ?", new String[] { name });
    return true;
```

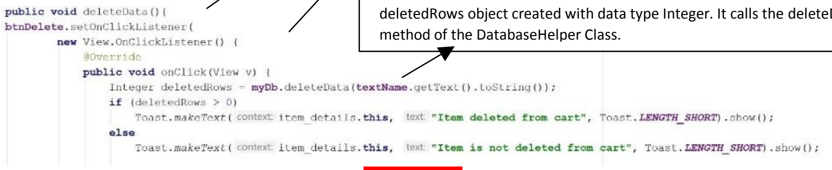
Fig. 32

'db.update' method. It takes three parameters: first, the Table in which the update is to be made; second, the contentValues where the changes is to be set; third, the identification to which product the changes are to be made i.e. it is identified through the name of the uniform selected; fourth, the String variable used i.e. name.

Delete Cart

Delete method constructed to delete data from the cart. The deleteData method is constructed under OnClickListener method of ‘Delete Cart’ button.

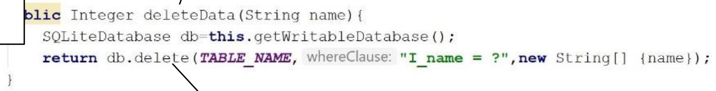
deletedRows object created with data type Integer. It calls the deleteData method of the DatabaseHelper Class.



```java
public void deleteData(){
btnDelete.setOnClickListener(
        new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Integer deletedRows = myDb.deleteData(textName.getText().toString());
                if (deletedRows > 0)
                    Toast.makeText( context: item_details.this,  text: "Item deleted from cart", Toast.LENGTH_SHORT).show();
                else
                    Toast.makeText( context: item_details.this,  text: "Item is not deleted from cart", Toast.LENGTH_SHORT).show();
```

Fig. 33

‘deleteData’ method constructed to delete an item selected in the list from the cart. It takes Integer return type which is the ‘row id’ in the database.



```java
blic Integer deleteData(String name){
  SQLiteDatabase db=this.getWritableDatabase();
  return db.delete(TABLE_NAME, whereClause: "I_name = ?",new String[] {name});
}
```

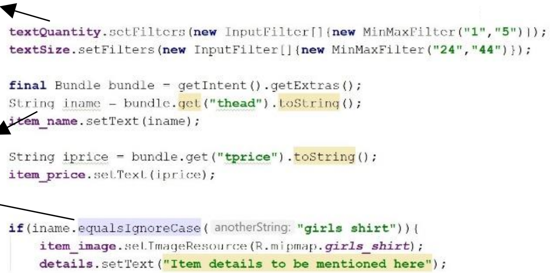
Fig. 34

The delete method takes three parameters: first, the Table name in the database where the delete action takes place’ second, the identification of the item which has to be deleted, it is identified by the name of the uniform; third, the String value used for the action i.e. ‘name.’ The row id of the database is identified by the name of the uniform

To set a limit on the quantity and size that can entered in the EditText object, ‘textQuantity’ and ‘textSize’. To set the Minimum and Maximum integer which can be entered the ‘MinMaxFilter’ Java Class is called (Refer to Fig. 33).

The text has been set to the TextView object where this data is to be displayed.

To set an image according to the item selected in the array list an ‘if-‘else’ method is used to compare the string values. Then, the image source has been set using ‘setImageResource’ method which takes the stored image as a parameter. According to the uniform item selected the information related to it like its material type and color would be mentioned in the details box (Refer to Fig. 19).



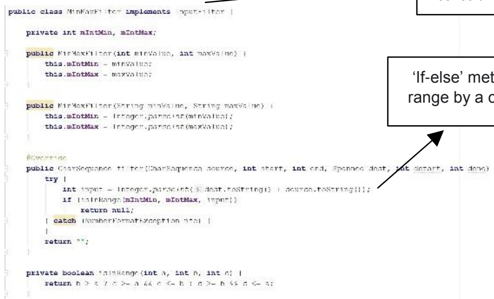
```java
textQuantity.setFilters(new InputFilter[]{new MinMaxFilter("1","5")});
textSize.setFilters(new InputFilter[]{new MinMaxFilter("24","44")});

final Bundle bundle = getIntent().getExtras();
String iname = bundle.get("thead").toString();
item_name.setText(iname);

String iprice = bundle.get("tprice").toString();
item_price.setText(iprice);


if(iname.equalsIgnoreCase( anotherString: "girls shirt")){
    item_image.setImageResource(R.mipmap.girls_shirt);
    details.setText("Item details to be mentioned here");
```

Fig. 35

Java Class: MinMaxFilter

The inputFilter is the interface attached to EditText to constraint the input type that can be made in it (EditText).

‘If-else’ method is used to check the number input by the user is in the range by a call to ‘isInRange’ method.



```java
public class MinMaxFilter implements InputFilter {

    private int mIntMin, mIntMax;

    public MinMaxFilter(int minValue, int maxValue) {
        this.mIntMin = minValue;
        this.mIntMax = maxValue;
    }

    public MinMaxFilter(String minValue, String maxValue) {
        this.mIntMin = Integer.parseInt(minValue);
        this.mIntMax = Integer.parseInt(maxValue);
    }

    @Override
    public CharSequence filter(CharSequence source, int start, int end, Spanned dest, int dstart, int dend)
        try {
            int input = Integer.parseInt( s: dest.toString() + source.toString());
            if (isInRange(mIntMin, mIntMax, input))
                return null;
        } catch (NumberFormatException nfe) {
        }
        return "";
    }

    private boolean isInRange(int a, int b, int c) {
        return b > a ? c >= a && c <= b : c >= b && c <= a;
    }
```

Fig. 36

Evidence of Offline MySQLite Database

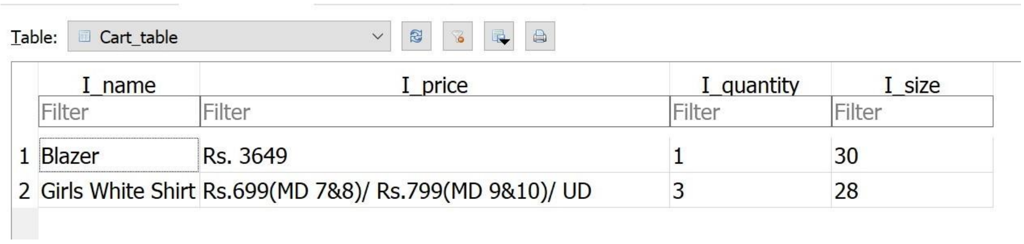


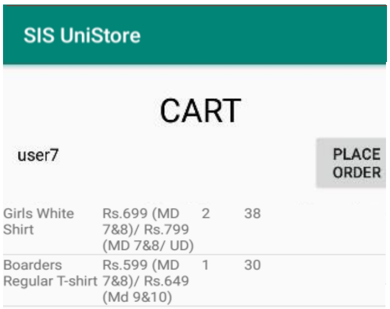
Table: Cart_table

	I_name	I_price	I_quantity	I_size
	Filter	Filter	Filter	Filter
1	Blazer	Rs. 3649	1	30
2	Girls White Shirt	Rs.699(MD 7&8)/ Rs.799(MD 9&10)/ UD	3	28

Fig. 25B

7. Cart Page

- The cart stores all the items that the user has selected to buy.
- It will use ListView to display this data. It also uses an adapter class with this.
- It also has a button to place the order. On clicking the button, an alert dialog box appears (Refer to Fig. 35).



SIS UniStore

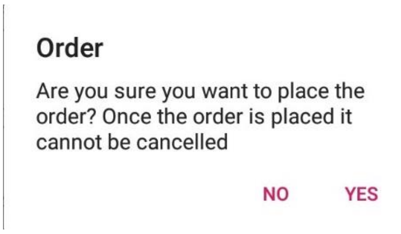
CART

user7

PLACE ORDER

Girls White Shirt	Rs.699 (MD 7&8)/ Rs.799 (MD 7&8/ UD)	2	38
Boarders Regular T-shirt	Rs.599 (MD 7&8)/ Rs.649 (Md 9&10)	1	30

Fig. 37 Fig. 6

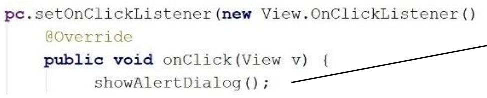


Order

Are you sure you want to place the order? Once the order is placed it cannot be cancelled

Fig. 38

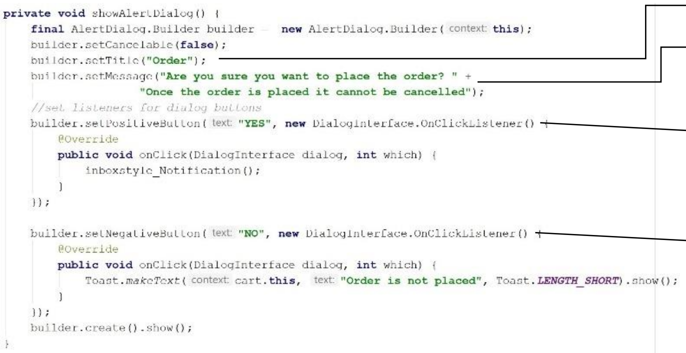
For the Alert Dialog Box 'showAlertDialogBox' method is constructed in 'Cart' Java Class.



```
pc.setOnClickListener(new View.OnClickListener()
    @Override
    public void onClick(View v) {
        showAlertDialog();
```

The alert dialog box appears when the 'Place Order' button is clicked (Refer to Fig. 34). Hence the 'showAlertDialog' method is called under the 'setOnClickListener' method.

Fig. 39



```
private void showAlertDialog() {
    final AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
    builder.setCancelable(false);
    builder.setTitle("Order");
    builder.setMessage("Are you sure you want to place the order? " +
                "Once the order is placed it cannot be cancelled");
    //set listeners for dialog buttons
    builder.setPositiveButton( text: "YES", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            inboxstyle_Notification();
        }
    });

    builder.setNegativeButton( text: "NO", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            Toast.makeText( context: cart.this,  text: "Order is not placed", Toast.LENGTH_SHORT).show();
        }
    });
    builder.create().show();
}
```

'setTitle' method sets the title of the Alert Dialog Box i.e. Order (Refer to Fig. 35).

'setMessage' method sets the message to be displayed in the Alert Dialog Box (Refer to Fig.35).

There are two buttons displayed: 'Yes' and 'No'.

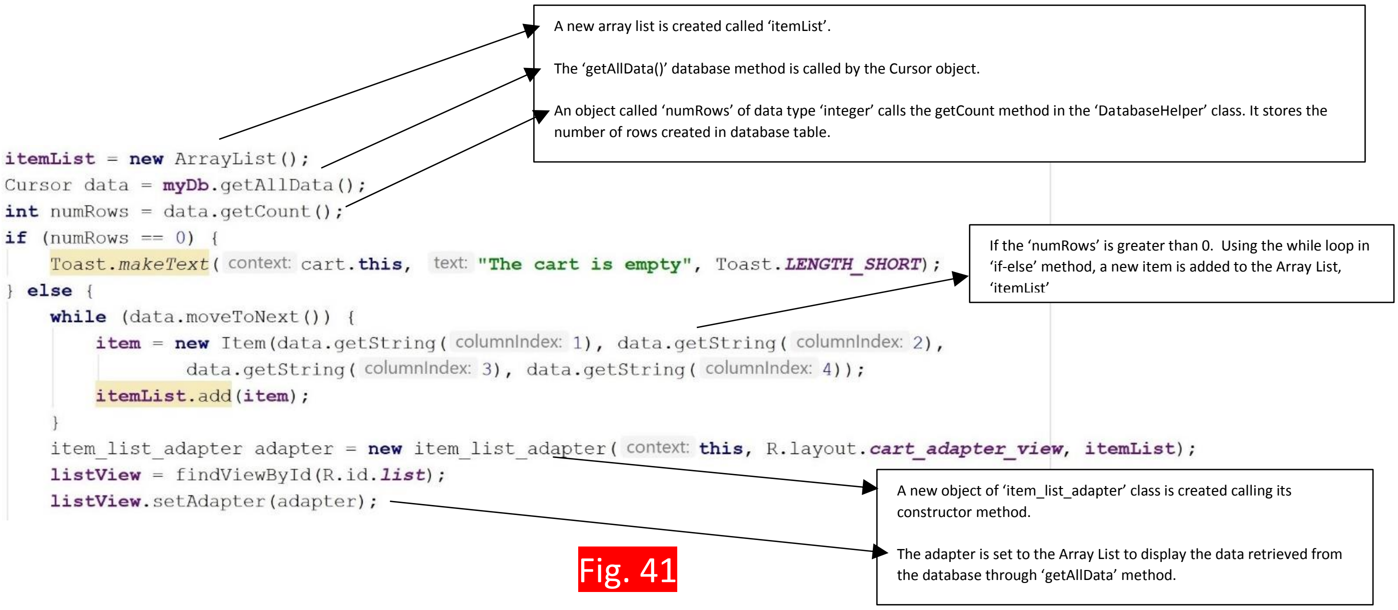
The 'setPositiveButton' method takes two parameters: first, the text to be displayed; second, the action to be taken on the text displayed. To make the text clickable Click Listener method is set. By pressing the 'Yes' button a toast message is displayed, and an Intent method makes it go to another Activity.

The 'setNegativeButton' method takes the same parameters as 'setPositiveButton'. Here only a Toast message is displayed.

Fig. 40

Java Class: Cart

- The View Cart Button created in ‘Item Details’ page views the whole cart. This is the Java Class of that Activity.



A new array list is created called ‘itemList’.

The ‘getAllData()’ database method is called by the Cursor object.

An object called ‘numRows’ of data type ‘integer’ calls the getCount method in the ‘DatabaseHelper’ class. It stores the number of rows created in database table.

```
itemList = new ArrayList();
Cursor data = myDb.getAllData();
int numRows = data.getCount();
if (numRows == 0) {
    Toast.makeText( context: cart.this,  text: "The cart is empty", Toast.LENGTH_SHORT);
} else {
    while (data.moveToNext()) {
        item = new Item(data.getString( columnIndex: 1), data.getString( columnIndex: 2),
                data.getString( columnIndex: 3), data.getString( columnIndex: 4));
        itemList.add(item);
    }
    item_list_adapter adapter = new item_list_adapter( context: this, R.layout.cart_adapter_view, itemList);
    listView = findViewById(R.id.list);
    listView.setAdapter(adapter);
```

If the ‘numRows’ is greater than 0. Using the while loop in ‘if-else’ method, a new item is added to the Array List, ‘itemList’

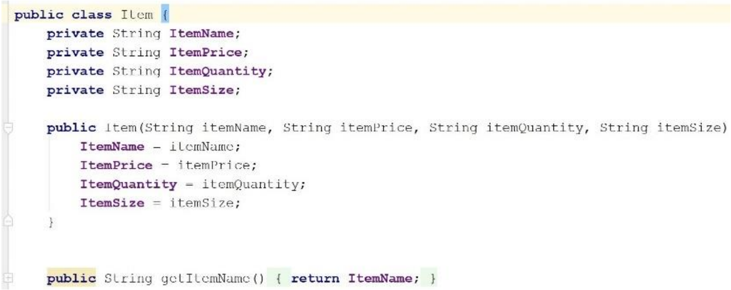
A new object of ‘item_list_adapter’ class is created calling its constructor method.

The adapter is set to the Array List to display the data retrieved from the database through ‘getAllData’ method.

Fig. 41

Java Class: Item

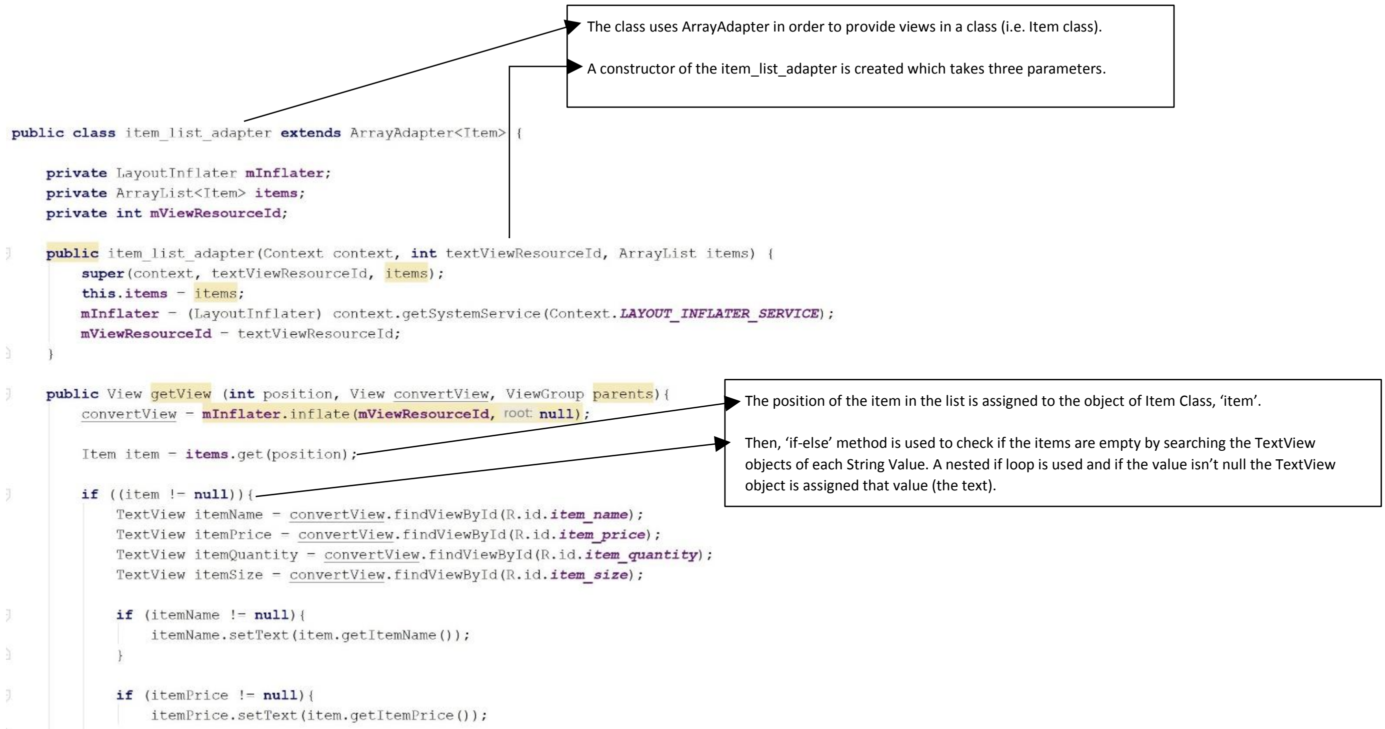
- To store all the data retrieved from the database, string values are created in the Item class.
- Accessor methods are defined to each string value to be used in other activities.



```
public class Item {
    private String ItemName;
    private String ItemPrice;
    private String ItemQuantity;
    private String ItemSize;

    public Item(String itemName, String itemPrice, String itemQuantity, String itemSize)
        ItemName = itemName;
        ItemPrice = itemPrice;
        ItemQuantity = itemQuantity;
        ItemSize = itemSize;
    }


    public String getItemName() { return ItemName; }
```

Fig. 42

Java Class: item_list_adapter



The class uses ArrayAdapter in order to provide views in a class (i.e. Item class).

A constructor of the item_list_adapter is created which takes three parameters.

```java
public class item_list_adapter extends ArrayAdapter<Item> {

    private LayoutInflater mInflater;
    private ArrayList<Item> items;
    private int mViewResourceId;

    public item_list_adapter(Context context, int textViewResourceId, ArrayList items) {
        super(context, textViewResourceId, items);
        this.items = items;
        mInflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        mViewResourceId = textViewResourceId;
    }

    public View getView (int position, View convertView, ViewGroup parents){
        convertView = mInflater.inflate(mViewResourceId, root: null);

        Item item = items.get(position);

        if ((item != null)){
            TextView itemName = convertView.findViewById(R.id.item_name);
            TextView itemPrice = convertView.findViewById(R.id.item_price);
            TextView itemQuantity = convertView.findViewById(R.id.item_quantity);
            TextView itemSize = convertView.findViewById(R.id.item_size);

            if (itemName != null){
                itemName.setText(item.getItemName());
            }

            if (itemPrice != null){
                itemPrice.setText(item.getItemPrice());
```

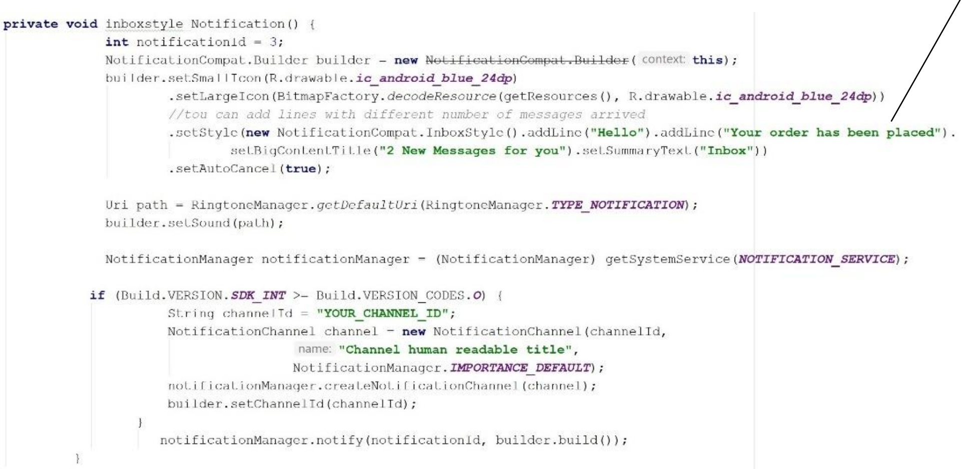
The position of the item in the list is assigned to the object of Item Class, 'item'.

Then, 'if-else' method is used to check if the items are empty by searching the TextView objects of each String Value. A nested if loop is used and if the value isn't null the TextView object is assigned that value (the text).

Fig. 43

8. Mail Notification method

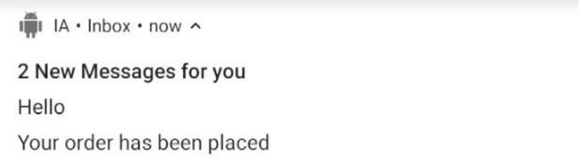
There are different types of notifications. This is the mail notification style.



```
private void inboxstyle Notification() {
            int notificationId = 3;
            NotificationCompat.Builder builder = new ~~NotificationCompat.Builder~~( context: this);
            builder.setSmallIcon(R.drawable.ic_android_blue_24dp)
                    .setLargeIcon(BitmapFactory.decodeResource(getResources(), R.drawable.ic_android_blue_24dp))
                    //tou can add lines with different number of messages arrived
                    .setStyle(new NotificationCompat.InboxStyle().addLine("Hello").addLine("Your order has been placed").
                            setBigContentTitle("2 New Messages for you").setSummaryText("Inbox"))
                    .setAutoCancel(true);

            Uri path = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
            builder.setSound(path);

            NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

          if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                    String channelId = "YOUR_CHANNEL_ID";
                    NotificationChannel channel = new NotificationChannel(channelId,
                                    name: "Channel human readable title",
                                    NotificationManager.IMPORTANCE_DEFAULT);
                    notificationManager.createNotificationChannel(channel);
                    builder.setChannelId(channelId);
                }
                   notificationManager.notify(notificationId, builder.build());
        }
```

Sets the written content of the notification that pops up. It may have changed in the process of fine-tuning.

Fig. 44



IA • Inbox • now ^

2 New Messages for you

Hello

Your order has been placed

Fig. 45

References

Firebase:

https://www.simplifiedcoding.net/android-firebase-tutorial-1/

My SQLite Database:

https://dzone.com/articles/create-a-database-android-application-in-android-s

https://www.tutorialspoint.com/android/android_sqlite_database.htm

https://youtu.be/cp2rL3sAFmI

Google Play Store

https://play.google.com/store/apps/details?id=com.hussainlabs.learnandroidapp