Criterion C: Development

1. Java Android/Program

1.1. Android Components

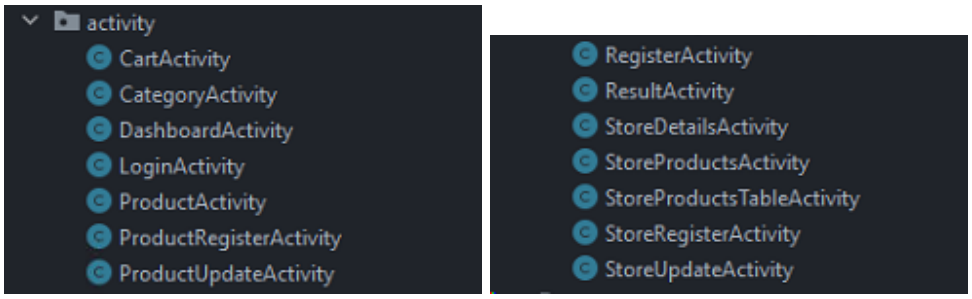
Activity

(androidx.appcompat.app.AppCompatActivity)

As part of Android development, Activity is an integral part of android apps. Each application has different pages known as Activity. In this program, several activities are created by inheriting and Overriding the AppCompatActivity Class to perform the functionality and success criteria. Below shows some activity classes in the program.

Figure 1

Program Activity Classes



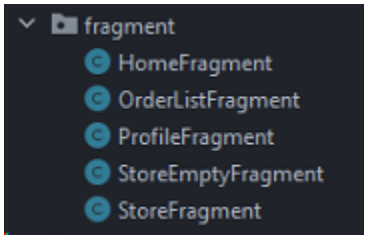
Fragments

(androidx.fragment.app.Fragment)

Fragments are pages that can be attached to certain Activities. Fragments allow for cycling of different pages without having to initiate new Activity. Each fragment needs to be imported to the activity and assigned to the current view of the Activity. Below shows lists of fragments for the DashboardActivity Class.

Figure 2

Program Fragment Classes



Bottom Navigation Menu

(com.google.android.material.bottomnavigation.BottomNavigationView)

BottomNavigationView Class is used to create a bottom navigation menu in the DashboardActivity. The navigation menu is used to navigate through the different dashboard fragments. For the BottomNavigationView to function, a list of menu items is required and attached to its XML element. The switch statement to change the fragment is shown below.

Figure 3

Bottom Navigation Menu Switch

```
// Method to check if a menu item is selected
public boolean onNavigationItemSelected(@NonNull MenuItem item) {
    // Switching between different fragments in dashboard when the menu item is pressed
    switch (item.getItemId()) {
        case R.id.nav_home_selector:
            getSupportFragmentManager().beginTransaction()
                    .replace(R.id.container, homeFragment).commit();
            return true;

        case R.id.nav_profile_selector:

            getSupportFragmentManager().beginTransaction()
                    .replace(R.id.container, profileFragment).commit();
            return true;

        case R.id.nav_store_selector:
            if (StoreModel.getStoreId() == null) {
                getSupportFragmentManager().beginTransaction()
                        .replace(R.id.container, storeEmptyFragment).commit();
            } else {
                getSupportFragmentManager().beginTransaction()
                        .replace(R.id.container, storeFragment).commit();
            }
            return true;
    }
    return false;
}
```

RecyclerView

(androidx.recyclerview.widget.RecyclerView)

The RecyclerView Class is used to create an infinite scroll list of items. RecyclerView requires an adapter to tell the current view of the RecyclerView to display items. In StoreProducts Activity, it is used to create a list of ten items then when the user scrolls to the bottom of the page, the next ten items are loaded. This class is also used in ProductCategory Activity. The sample code of the *StoreProductActivity* implementing the RecyclerView is shown below along with the *RecyclerViewAdapter*.

Figure 4

StoreProductActivity Class

```
package com.kenazalfizza.farmersmarketplace.store;

import ...

public class StoreProductsActivity extends AppCompatActivity {
    // Variable declaration
    RecyclerView recyclerView;
    RecyclerViewProductListAdapter recyclerViewProductListAdapter;
    ArrayList<ProductListItem> rowsArrayProductList = new ArrayList<>();

    boolean isLoading = false;
    boolean allLoaded = false;

    ImageButton btn_goto_product_register;
    static String storeId;
    String baseURL = "http://192.168.1.5/";

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    public static void setStoreId(String storeId) { StoreProductsActivity.storeId = storeId; }

    @Override
    protected void onResume() {
        super.onResume();
        getProductList();
        initAdapter(UserStoreProductsCurrent.getProductListItems());
    }

    protected void elementsFindViewById() {...}

    private void getProductList() {...}

    private void initAdapter(ArrayList<ProductListItem> rowsArrayProductList) {
        // Initialising the adapter for recyclerView
        recyclerViewProductListAdapter = new RecyclerViewProductListAdapter(rowsArrayProductList, baseURL);
        recyclerView.setAdapter(recyclerViewProductListAdapter);
    }
}
```

Figure 5

RecyclerViewProductListAdapter Class

```java
package com.kenazalfizza.farmersmarketplace.view;

import ...

public class RecyclerViewProductListAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {

    private final int VIEW_TYPE_ITEM = 0;
    private final int VIEW_TYPE_LOADING = 1;
    String baseURL;

    // Array declaration to store the data of each item
    public List<ProductListItem> mItemList;

    public RecyclerViewProductListAdapter(List<ProductListItem> itemList, String baseURL) {
        // Assigning the value of array
        mItemList = itemList;
        this.baseURL = baseURL;

    }

    @NonNull
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        // Check if the view type item of the lists is a ViewItem
        if (viewType == VIEW_TYPE_ITEM) {
            // If true
            View view = LayoutInflater.from(parent.getContext())
                    .inflate(R.layout.list_product_item, parent, attachToRoot: false);
            // Inflating the view with layout of R.layout.list_product_item
            return new ItemViewHolder(view);
            // Returning the view of ItemViewHolder
        } else {
            // If false
            View view = LayoutInflater.from(parent.getContext())
                    .inflate(R.layout.progress, parent, attachToRoot: false);
            // Inflating the view with layout of R.layout.progress
            return new LoadingViewHolder(view);
            // Return the view of LoadingViewHolder
        }
    }
```

```
62
63      @Override
64      public void onBindViewHolder(@NonNull RecyclerView.ViewHolder viewHolder, int position) {
65
66          // When the program is binding its viewHolder
67          if (viewHolder instanceof ItemViewHolder) {
68              // If the instance bind is a Recycler.ViewHolder object, it will populate the lists
69              populateItemRows((ItemViewHolder) viewHolder, position);
70          } else if (viewHolder instanceof LoadingViewHolder) {
71              // If the instance bind is a LoadingViewHolder object, it will show loading view
72              showLoadingView((LoadingViewHolder) viewHolder, position);
73          }
74
75      }
76
77      @Override
78      public int getItemCount() { return mItemList == null ? 0 : mItemList.size(); }
81
82      /**
83       * The following method decides the type of ViewHolder to display in the RecyclerView
84       *
85       * @param position
86       * @return
87       */
88      @Override
89      public int getItemViewType(int position) {
90          return mItemList.get(position) == null ? VIEW_TYPE_LOADING : VIEW_TYPE_ITEM;
91      }
92
93
94      private class ItemViewHolder extends RecyclerView.ViewHolder {
95
96          String id;
97          // Declare the view element to be changed here
98          TextView tvItem;
99          TextView tvPrice;
100         TextView tvLocation;
101         CardView cvItemList;
102         ImageView ivImage;
```

```
104         public ItemViewHolder(@NonNull View itemView) {
105             super(itemView);
106
107             // Finding the view element by id
108             tvItem = itemView.findViewById(R.id.item);
109             tvPrice = itemView.findViewById(R.id.price);
110             tvLocation = itemView.findViewById(R.id.location);
111
112             cvItemList = itemView.findViewById(R.id.cv_item_list);
113
114             ivImage = itemView.findViewById(R.id.image);
```

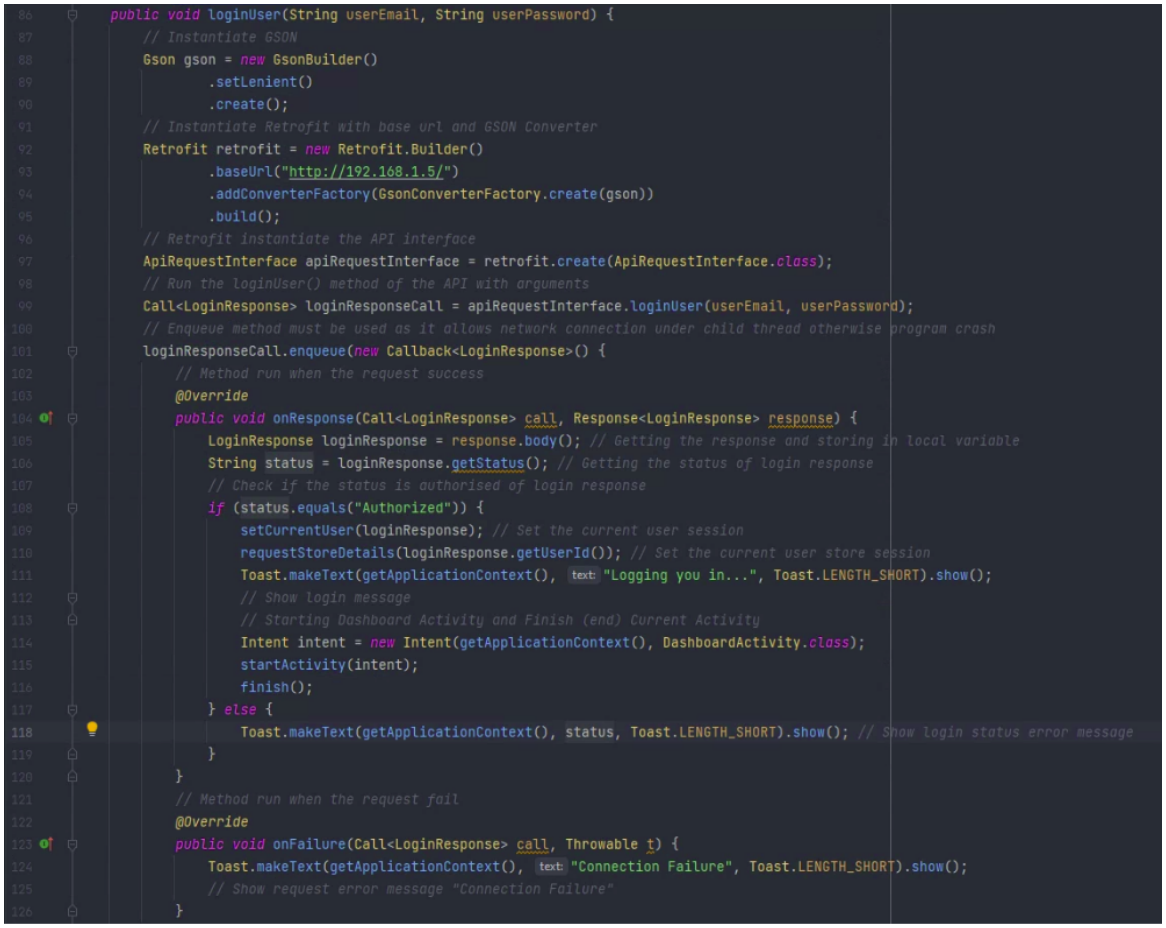
```
            cvItemList.setOnClickListener(new View.OnClickListener() {
                // Method when the user click the card item of the lists
                @Override
                public void onClick(View view) {
                    Context context = view.getContext();
                    Intent intent = new Intent(context, ProductActivity.class);
                    ProductActivity.setProductId(id);
                    context.startActivity(intent);
                    // Starting ProductActivity page
                }
            });

        }

    }
```

```
    private class LoadingViewHolder extends RecyclerView.ViewHolder {

        // Initialising progressBar
        ProgressBar progressBar;

        public LoadingViewHolder(@NonNull View itemView) {
            super(itemView);
            // finding view of progressBar by id
            progressBar = itemView.findViewById(R.id.progressBar);
        }
    }

    private void showLoadingView(LoadingViewHolder viewHolder, int position) {
        //ProgressBar would be displayed when the page is loading more item

    }
    @SuppressLint("SetTextI18n")
    private void populateItemRows(ItemViewHolder viewHolder, int position) {
        Log.d( tag: null, mItemList.toString());
        // Getting the element of the arrays at index
        // corresponding position to the item on the lists
        ProductListItem productListItemItem = mItemList.get(position);

        // Setting the element of the list items
        viewHolder.id = productListItemItem.getId();
        viewHolder.tvItem.setText(productListItemItem.getName());
        viewHolder.tvLocation.setText(productListItemItem.getLocation());
        viewHolder.tvPrice.setText("Rp" + productListItemItem.getPrice());
        //viewHolder.ivImage.setImageBitmap(image);

        String resURL = baseURL + "marketplace/product/res/" + productListItemItem.getId() + ".png";
        Picasso.get() Picasso
                .load(resURL) RequestCreator
                .fit()
                .centerCrop()
                .into(viewHolder.ivImage);
    }
}
```

1.2. Java Techniques

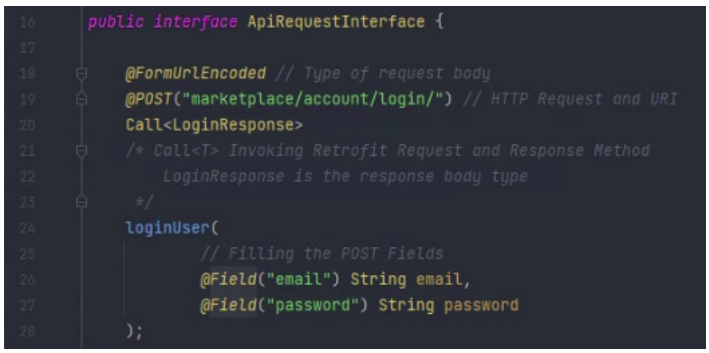
Retrofit & GSON

Retrofit is a REST API handling the transfer of data to and from a website. GSON is a library that allows parsing JSON Objects to and from Java Objects. Both are implemented to handle data transfer between webserver and the application. The usage of Retrofit in the login page is shown to request login to the web server.



```
public void loginUser(String userEmail, String userPassword) {
    // Instantiate GSON
    Gson gson = new GsonBuilder()
            .setLenient()
            .create();
    // Instantiate Retrofit with base url and GSON Converter
    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://192.168.1.5/")
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build();
    // Retrofit instantiate the API interface
    ApiRequestInterface apiRequestInterface = retrofit.create(ApiRequestInterface.class);
    // Run the loginUser() method of the API with arguments
    Call<LoginResponse> loginResponseCall = apiRequestInterface.loginUser(userEmail, userPassword);
    // Enqueue method must be used as it allows network connection under child thread otherwise program crash
    loginResponseCall.enqueue(new Callback<LoginResponse>() {
        // Method run when the request success
        @Override
        public void onResponse(Call<LoginResponse> call, Response<LoginResponse> response) {
            LoginResponse loginResponse = response.body(); // Getting the response and storing in local variable
            String status = loginResponse.getStatus(); // Getting the status of login response
            // Check if the status is authorised of login response
            if (status.equals("Authorized")) {
                setCurrentUser(loginResponse); // Set the current user session
                requestStoreDetails(loginResponse.getUserId()); // Set the current user store session
                Toast.makeText(getApplicationContext(), text: "Logging you in...", Toast.LENGTH_SHORT).show();
                // Show login message
                // Starting Dashboard Activity and Finish (end) Current Activity
                Intent intent = new Intent(getApplicationContext(), DashboardActivity.class);
                startActivity(intent);
                finish();
            } else {
                Toast.makeText(getApplicationContext(), status, Toast.LENGTH_SHORT).show(); // Show login status error message
            }
        }
        // Method run when the request fail
        @Override
        public void onFailure(Call<LoginResponse> call, Throwable t) {
            Toast.makeText(getApplicationContext(), text: "Connection Failure", Toast.LENGTH_SHORT).show();
            // Show request error message "Connection Failure"
        }
```

Front the code snippet, A Java Interface (*ApiRequestInteface* Class) contains different server requests and responses instantiated. Below is the loginUser interface used in the login page.



```
public interface ApiRequestInterface {

    @FormUrlEncoded // Type of request body
    @POST("marketplace/account/login/") // HTTP Request and URI
    Call<LoginResponse>
    /* Call<T> Invoking Retrofit Request and Response Method
        LoginResponse is the response body type
     */
    loginUser(
            // Filling the POST Fields
            @Field("email") String email,
            @Field("password") String password
    );
```

Response class is responsible for containing the response body. Sample Response Class, *LoginResponse*

```java
public class LoginResponse {
    @SerializedName("status")
    String status;

    @SerializedName("id")
    String userId;

    @SerializedName("name")
    String userName;

    @SerializedName("email")
    String userEmail;

    @SerializedName("phone")
    String userPhone;

    public LoginResponse(String userId, String userName, String userEmail, String userPhone) {
        this.userId = userId;
        this.userName = userName;
        this.userEmail = userEmail;
        this.userPhone = userPhone;
    }

    public String getStatus() { return status; }

    public String getUserId() { return userId; }

    public void setUserId(String userId) { this.userId = userId; }

    public String getUserName() { return userName; }

    public void setUserName(String userName) { this.userName = userName; }

    public String getUserEmail() { return userEmail; }

    public void setUserEmail(String userEmail) { this.userEmail = userEmail; }

    public String getUserPhone() { return userPhone; }

    public void setUserPhone(String userPhone) { this.userPhone = userPhone; }
```

In order for GSON to recognise which JSON variable' values assigned to which Object variable' values, *@SerializedName* annotation is required on variables. Steers and Getters are used to set the current user session.

Cryptography

In this program encryption method is used to encrypt the password of the user accounts before it is sent to the database for security purposes. The figure 6 shows CryptoHash Class used to encrypt the Strings using the SHA-256 public key.

Figure 6

CryptoHash Class used to generate hashed texts

```java
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.nio.charset.StandardCharsets;
import java.math.BigInteger;

public class CryptoHash {
    private final String text;

    public CryptoHash(String text) { this.text = text; }

    public String run() {
        MessageDigest md = null;
        try {
            md = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }

        md.update(text.getBytes(StandardCharsets.UTF_8));
        byte[] digest = md.digest();

        String hex = String.format("%064x", new BigInteger( signum: 1, digest));
        return hex;
    }
}
```

Iterative Object Loops

Continuing from the previous section, the array of Product[] products array is looped through iteratively using for loops. This is required to obtain the value of variables in each object in the array. Below shows the code for the loop during a retrieval of product list.

Figure 7

Iterative Object Loop Sample

```java
for (Product product : products) {
    rowsArrayIdList.add(product.getProductId());
    rowsArrayItemList.add(product.getProductName());
    rowsArrayLocationList.add(product.getStoreLocation());
    rowsArrayPriceList.add(String.valueOf(product.getProductPrice()));
}
```

ArrayList

ArrayList is used instead of arrays as it allows for variable length once the array is created. This is used in creation of product list and cart list which varies depending on the stored data in the database.

HashMap

HashMap allows the assignment of pairs of keys with values which is useful in combination with ArrayList to create a list of objects. In the program this is used to store key and values when creating list of products and list of products carts

2. Web Server

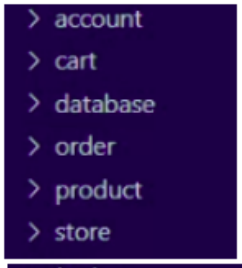
XAMPP is the tool used to run and manage the database and webserver. This tool is used to ease the initialization of database and web servers, as the development focuses on the program, web server, and database administration.

Web Server PHP and Directories

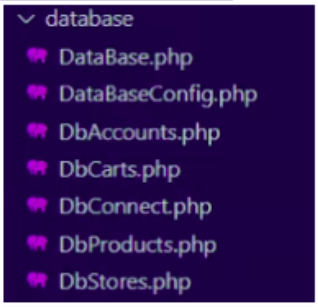
The web server consists of several URLs which handle request and response applications and databases. Depending on the function currently requested by the program, a certain URL will be the target.

Figure 8

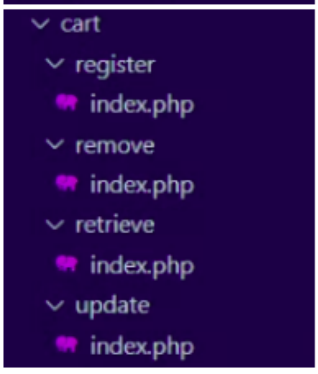
Image of storeRegister.php as a sample API which imports the *DbStores.php*, instantiate the *Stores* Class, and calling the function *storeRegister()*



The functionalities of the web server in the target URL handle communication of data that includes user, store, product, carts, orders, etc. The different functions are split into directories below.



Inside the database folder consists of PHP files of database configuration and connection and Db php files containing functions to handle different requests. The file that contains the name “Db..” consists of the class, functions, and constructors of their respective API category. This file will then be included to the other php files and the class inside will be instantiated.



In the other directory, each folder contains the index.php files under directory functions. The index.php file handles the request from the program which the program accesses and executes the certain functions from the “Db…” files. Figure 9 shows the index.php sample

Figure 9

Image of index.php of store register which imports the DbStores.php, instantiate the Stores Class, and calling the function storeRegister() and echo the result

store > register > index.php

```php
<?php
require '../../database/DbStores.php'; // Importing 'DbStores.php' file
$db = new Stores(); // Instantiation of Stores Class

if ($db->dbConnect()) {
    $result = $db->storeRegister($_POST['user_id'],
                                 $_POST['store_id'],
                                 $_FILES['store_image'],
                                 $_POST['store_name'],
                                 $_POST['store_email'],
                                 $_POST['store_phone'],
                                 $_POST['store_province'],
                                 $_POST['store_city'],
                                 $_POST['store_postcode'],
                                 $_POST['store_address'],
                                 $_POST['store_description']);
        // $db->storeRegister() is calling the function of
        // instantiated $db object based on the Stores Class
} else $result =  "Error: Database Connection";

header('Content-Type: application/json');
echo json_encode($result);

?>
```

HTTP POST Requests

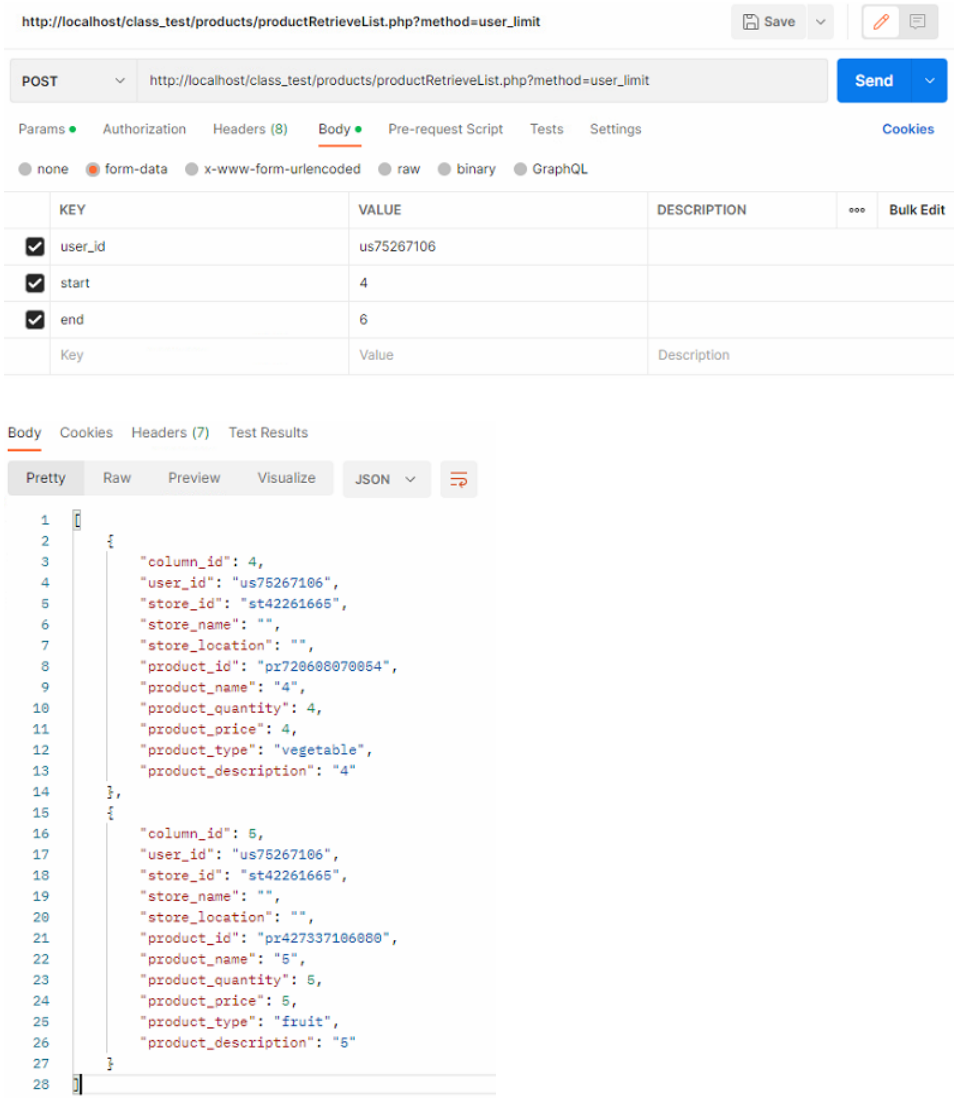
In the webserver, $_POST is used as a form required in the web server depending on the performed operations. The web server accepts $_POST mainly to obtain the value of variables passed as a parameter to the function called by the php page. The previous figure, the $_POST values are passed as a parameter to the *storeRegister()* method. This method I applicable with other parts of the webserver

JSON Object and Array to transfer data between platform

In this ecosystem, the web server sends JSON Object Array to the program. JSON is used to send data between platforms as it is a data-interchange format and it is easy to parse by the program. The JSON Object Array sample as a result from visiting a php file is shown below

Figure 13

Image of sending request to php location with $_POST keys and values and the JSON Object Array Result



Note: The tool used to send request to a webpage in this figure is Postman v10.5.2

3. Database

PhpMyAdmin is a tool used to manage databases for this program. The tables created for the application database are shown below.

Figure 15

Image of database tables for the program

	Table	Action	Rows	Type	Collation	Size	Overhead
☐	carts	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	cart_products	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	cart_stores	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	communities	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	community_members	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	images	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	orders	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	orders_status	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	products	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	64.0 KiB	-
☐	stores	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	users	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_general_ci	48.0 KiB	-

Appendix

Appendix F.1. StoreFragment Class

```
public class StoreFragment extends Fragment {

    TextView tv_storeName, tv_storePhone, tv_storeLocation;
    LinearLayout li_gotoStoreDetails, li_gotoStoreProducts;
    LinearLayout li_productTest;
    StoreEmptyFragment storeEmptyFragment = new StoreEmptyFragment();

    public StoreFragment() {}

    @Override
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); }

    @Override
    public void onResume() {
        super.onResume();
        elementsSetDetails();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view = inflater.inflate(R.layout.fragment_store, container, attachToRoot: false);
        elementsFindViewById(view);
        elementsSetDetails();
        li_gotoStoreDetails.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(getActivity(), StoreDetailsActivity.class);
                startActivity(intent);
            }
        });
        li_gotoStoreProducts.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(getActivity(), StoreProductsActivity.class);
                startActivity(intent);
            }
        });
        li_productTest.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ProductActivity.setProductId("pr816544307671");
                Intent intent = new Intent(getActivity(), ProductActivity.class);
                startActivity(intent);
            }
        });
        return view;
    }
```

```
    public void elementsFindViewById(View view) {
        // Find View By Id for TextView
        tv_storeName = (TextView) view.findViewById(R.id.tv_store_name);
        tv_storePhone = (TextView) view.findViewById(R.id.tv_store_phone);
        tv_storeLocation = (TextView) view.findViewById(R.id.tv_store_location);
        li_gotoStoreDetails = (LinearLayout) view.findViewById(R.id.li_goto_store_details);
        li_gotoStoreProducts = (LinearLayout) view.findViewById(R.id.li_goto_store_products);
        li_productTest = (LinearLayout) view.findViewById(R.id.li_product_test);
    }

    public void elementsSetDetails() {
        tv_storeName.setText(StoreModel.getStoreName());
        tv_storePhone.setText(StoreModel.getStorePhone());
        tv_storeLocation. setText(StoreModel.getStoreCity());
    }
}
```