Lab Report: Dynamic Flowers List Application with Image Visualization

TITLE

Lab Report: Understanding and Implementing ListView in Android for a Dynamic Flowers List Application

INTRODUCTION

This report details the theoretical concepts and practical implementation of ListView within an Android mobile application designed to manage a dynamic list of flowers. The application allows users to add and delete flower names, visualize associated images directly within the list, and view larger images and details on a separate screen. This exercise serves to illustrate fundamental Android UI development patterns, data management techniques, and inter-activity communication.

OBJECTIVES

The objectives of this lab report are:

- To understand the core concepts of ListView and its role in displaying dynamic lists in Android.
- To explain the necessity and function of Adapters (specifically ArrayAdapter) in populating a ListView with custom item layouts.
- To demonstrate how to define custom layouts for individual list items, including both text and images.
- To illustrate the implementation of dynamic adding and deleting of flower names from the list.
- To describe how ListView item clicks can trigger navigation to a separate detail activity, passing relevant data (flower name) for display.
- To utilize attractive UI design principles, including CardView and custom drawables, to enhance user experience.

PROCEDURE / ANALYSIS / DESIGN

Analysis of ListView

ListView is a UI component used for displaying a vertically scrolling collection of items. It is efficient in handling large datasets by recycling views, which means views that scroll off-screen are reused for new items that scroll into view, reducing memory consumption and improving performance. For this application, ListView was chosen to

present the dynamic collection of flowers.

Design Considerations

The "Flowers List" application was designed with two main screens: a list view and a detail view.

1. **Main List Screen (activity_main.xml):**
    - **Layout:** A LinearLayout serves as the root container, arranging elements vertically.
    - **Title:** A prominent TextView displays "My Flowers List."
    - **Input Section:** A horizontal LinearLayout contains an EditText (etFlowerName) for entering flower names and a Button (btnAddFlower) for adding them. This section is visually grouped with padding, a rounded background (@drawable/rounded_list_bg), and a subtle shadow (android:elevation).
    - **List Display:** A ListView (listViewFlowers) is used to show the list of flowers. It is aesthetically enhanced by being nested within a androidx.cardview.widget.CardView, providing rounded corners and a distinct elevated appearance. The ListView uses transparent dividers with increased height (android:dividerHeight="12dp") for better visual separation between items.
2. **List Item Layout (list_item_flower.xml):** Each item in the ListView is a custom layout.
    - It's a horizontal LinearLayout that holds an ImageView (ivFlowerIcon) for the flower's picture, a TextView (tvFlowerNameItem) for its name, and a Button (btnDeleteItem) for removal.
    - Custom drawables like rounded_list_item_bg are applied for a consistent, attractive look.
3. **Detail Screen (activity_flower_detail.xml):**
    - A simple vertical LinearLayout displays a large TextView (tvDetailFlowerName) for the flower's name and a large ImageView (ivDetailFlowerImage) for its full picture. The ImageView also uses a rounded background for visual appeal.
4. **Data Management:**
    - An ArrayList<String> (flowerList) stores the names of the flowers added by the user.
    - A HashMap<String, Integer> (flowerImageMap) is used to store the mapping between lowercase flower names (e.g., "rose") and their corresponding drawable resource IDs (e.g., R.drawable.rose). This allows for dynamic image loading based on the entered name.

5. **Navigation:** Intents are used to transition from MainActivity to FlowerDetailActivity, passing the selected flower's name.

IMPLEMENTATION

activity_main.xml (Main List Screen Layout)

The provided activity-main-flowers-list-xml immersive artifact defines the layout. Key elements include:

- Root LinearLayout with android:padding="24dp" and android:background="#e0f2f7".
- Input LinearLayout with android:padding="12dp", android:background="@drawable/rounded_list_bg", and android:elevation="4dp".
- EditText (@id/etFlowerName) and Button (@id/btnAddFlower) for adding flowers.
- ListView (@id/listViewFlowers) wrapped in a androidx.cardview.widget.CardView with app:cardCornerRadius="12dp" and app:cardElevation="6dp".

list_item_flower.xml (Custom List Item Layout)

This layout defines each row in the ListView. It includes an ImageView (@id/ivFlowerIcon), a TextView (@id/tvFlowerNameItem), and a Button (@id/btnDeleteItem). Styling uses rounded_list_item_bg and gradient_button_secondary.

MainActivity.java (Main List Screen Logic)

The core logic for the main list screen is handled in MainActivity.java.

- **Initialization of UI Components and Data Structures:**

```
public class MainActivity extends AppCompatActivity {
    private EditText etFlowerName;
    private Button btnAddFlower;
    private ListView listViewFlowers;
    private List<String> flowerList;
    private FlowerAdapter flowerAdapter;
    private Map<String, Integer> flowerImageMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

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

```
        btnAddFlower = findViewById(R.id.btnAddFlower);
        listViewFlowers = findViewById(R.id.listViewFlowers);

        flowerList = new ArrayList<>();
        flowerImageMap = new HashMap<>();
        // Populate flowerImageMap (e.g., flowerImageMap.put("rose",
R.drawable.rose);)

        flowerAdapter = new FlowerAdapter(this, R.layout.list_item_flower,
flowerList);
        listViewFlowers.setAdapter(flowerAdapter);

        // ... (Listeners for Add button and ListView items) ...
    }
    // ... (addFlower method and FlowerAdapter class) ...
}
```

- Custom FlowerAdapter (getView() method for item display and listeners):
  This custom ArrayAdapter is crucial for displaying the image and text in each list item and handling the delete button.

```
private class FlowerAdapter extends ArrayAdapter<String> {
    // ... (constructor and other methods) ...

    @Override
    public View getView(final int position, View convertView, ViewGroup parent) {
        ViewHolder holder;
        if (convertView == null) {
            convertView = LayoutInflater.from(getContext()).inflate(layoutResource,
parent, false);
            holder = new ViewHolder();
            holder.ivFlowerIcon = convertView.findViewById(R.id.ivFlowerIcon);
            holder.tvFlowerNameItem =
convertView.findViewById(R.id.tvFlowerNameItem);
            holder.btnDeleteItem = convertView.findViewById(R.id.btnDeleteItem);
            convertView.setTag(holder);
        } else {
            holder = (ViewHolder) convertView.getTag();
        }
```

```
        final String flower = flowers.get(position);
        String displayFlowerName = flower.substring(0,
    1).toUpperCase(Locale.getDefault()) +
    flower.substring(1).toLowerCase(Locale.getDefault());
        holder.tvFlowerNameItem.setText(displayFlowerName);

        Integer imageResId = flowerImageMap.get(flower);
        if (imageResId != null) {
            holder.ivFlowerIcon.setImageResource(imageResId);
        } else {
            holder.ivFlowerIcon.setImageResource(R.drawable.placeholder_flower);
        }

        holder.btnDeleteItem.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                flowers.remove(position);
                notifyDataSetChanged();
                Toast.makeText(getContext(), displayFlowerName + " removed.",
    Toast.LENGTH_SHORT).show();
            }
        });
        return convertView;
    }

    private class ViewHolder { // Optimizes view recycling
        ImageView ivFlowerIcon;
        TextView tvFlowerNameItem;
        Button btnDeleteItem;
    }
}
```

- **Adding Flowers (btnAddFlower OnClickListener):**

```
btnAddFlower.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String flowerName = etFlowerName.getText().toString().trim();
        if (flowerName.isEmpty()) {
            Toast.makeText(MainActivity.this, "Please enter a flower name.",
```

```
Toast.LENGTH_SHORT).show();
            return;
        }
        String lowerCaseFlowerName =
flowerName.toLowerCase(Locale.getDefault());
        if (!flowerList.contains(lowerCaseFlowerName)) {
            flowerList.add(lowerCaseFlowerName);
            flowerAdapter.notifyDataSetChanged();
            etFlowerName.setText("");
            Toast.makeText(MainActivity.this, flowerName + " added to list.",
Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(MainActivity.this, flowerName + " is already in the list.",
Toast.LENGTH_SHORT).show();
        }
    }
});
```

- **Item Click for Navigation (listViewFlowers OnItemClickListener):**

```
listViewFlowers.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id)
{
        String selectedFlower = flowerList.get(position);
        Intent intent = new Intent(MainActivity.this, FlowerDetailActivity.class);
        intent.putExtra("FLOWER_NAME", selectedFlower); // Pass data to detail
activity
        startActivity(intent);
    }
});
```

activity_flower_detail.xml (Detail Screen Layout)

This layout displays the selected flower's details, featuring a TextView (@id/tvDetailFlowerName) for the name and a large ImageView (@id/ivDetailFlowerImage) for the picture, both styled with rounded backgrounds.

FlowerDetailActivity.java (Detail Screen Logic)

The FlowerDetailActivity.java handles displaying the details of the selected flower.

- **Initialization and Data Retrieval:**

```
public class FlowerDetailActivity extends AppCompatActivity {
    private TextView tvDetailFlowerName;
    private ImageView ivDetailFlowerImage;
    private Map<String, Integer> flowerImageMap; // Same map as in MainActivity

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_flower_detail);

        tvDetailFlowerName = findViewById(R.id.tvDetailFlowerName);
        ivDetailFlowerImage = findViewById(R.id.ivDetailFlowerImage);

        // Populate flowerImageMap again (must match MainActivity's map)
        flowerImageMap = new HashMap<>();
        flowerImageMap.put("rose", R.drawable.rose);
        // ... add all other flower mappings ...

        String flowerName = getIntent().getStringExtra("FLOWER_NAME"); // Retrieve
passed data

        if (flowerName != null && !flowerName.isEmpty()) {
            String displayFlowerName = flowerName.substring(0,
1).toUpperCase(Locale.getDefault()) +
flowerName.substring(1).toLowerCase(Locale.getDefault());
            tvDetailFlowerName.setText(displayFlowerName);

            Integer imageResId = flowerImageMap.get(flowerName);
            if (imageResId != null) {
                ivDetailFlowerImage.setImageResource(imageResId);
            } else {
                ivDetailFlowerImage.setImageResource(R.drawable.placeholder_flower);
            }
        } else {
            tvDetailFlowerName.setText("No Flower Selected");
            ivDetailFlowerImage.setImageResource(R.drawable.placeholder_flower);
        }
    }
```

```
}
```

Drawable Resources

Various XML drawable files (rounded_edittext_bg.xml, gradient_button_primary.xml, gradient_button_secondary.xml, rounded_list_bg.xml, rounded_list_item_bg.xml, placeholder_flower.xml) are used to provide consistent and attractive styling across the application's UI elements, including rounded corners, gradient buttons, and subtle shadows. Actual flower image files (e.g., rose.png, lily.jpg) are placed in res/drawable/.

CONCLUSION

This laboratory exercise successfully demonstrated the creation of a dynamic list application using ListView in Android, specifically for managing a list of flowers. The objectives of dynamically adding and deleting list elements, displaying images within the list items, and navigating to a detail screen with full image visualization were met. The implementation highlighted the importance of custom ArrayAdapter and the ViewHolder pattern for efficient list rendering. Furthermore, the application incorporated attractive UI design principles through the use of CardView and custom XML drawables, resulting in a visually appealing and user-friendly experience. This project reinforced core Android development concepts essential for building interactive and data-driven mobile applications.