Ganpat University | U.V. Patel College of Engineering

U.V. Patel College of Engineering

GANPAT UNIVERSITY, KHERVA-384012, DIST - MEHSANA. (N.G.)

Name:- Krishna Y. Joshi

Enrollment No:- 21012011036

Class:- CE-IT-B

Batch:- 5B1

Subject:- Mobile Application Development

Page No.

Assignment: 1

1) Based on your understanding, identify a recent business trend that has influenced the Android platform. Explain how this trend impacts Android app developers and businesses in the mobile app industry.

A recent business trend that has influenced the Android platform is the rise of the subscription economy. More and more consumers are willing to pay for monthly or annual subscriptions to access good services, including mobile apps. This trend has had a significant impact on Android app developers and business in the mobile app industry. For Android app developers it offers a number of advantages and some challenges also.

Here are some specific examples of how the subscription economy has impacted Android app developers and businesses in the mobile app industry:

→ Many popular Android apps, such as Spotify, Netflix, now offer subscription plans. This has allowed these apps to generate more revenue and to invest in new features and content.

→ Some Android app developers are now offering subscription plans for their premium apps. This allows developers to generate more revenue from their apps and to provide users with access to exclusive features and content.

→ Some businesses in the mobile app industry are now offering

subscription plans for their app bundles. This allows businesses to offer users access to a variety of apps for a single monthly fee.

2) What is the purpose of an Inflator of layout in Android development, and how does it fit into the architecture of Android layouts?

The purpose of an inflator of layout in Android development is to convert an XML layout file into a View hierarchy. The inflator takes an XML layout file as input and create a View object for each element in the file. The inflator also sets the attributes of each View object based on the XML file.
The inflator is a key component of the Android layout architecture. It is responsible for creating the View hierarchy that is displayed on the screen. The View hierarchy is a tree of View objects, with each View object representing a different element on the screen.

3) Explain the concept of a CustomDialogBox in Android applications. Provide examples to illustrate it.

A CustomDialogBox also known as a Custom Dialog or Custom Alert Dialog, is a user interface element that allows developers to create a custom popup dialog with their own layout, content, and functionality. This is useful when you need to display information, gather input, or perform actions in a way that doesn't fit the standard Android dialog styles.

activity_main.xml:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView
        android:id="@+id/dialogTitle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Custom Dialog"
        android:gravity="center"
        android:padding="16dp" />
    <Button
        android:id="@+id/closeButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Close"
        android:padding="16dp" />
</LinearLayout>
```

MainActivity.kt:

```
import android.app.Dialog
import android.os.Bundle
import android.view.View
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity
```

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val showDialogButton = findViewById<Button>(R.id.showDialogButton)
        showDialogButton.setOnClickListener {
            showCustomDialog()
        }
    }

    private fun showCustomDialog() {
        val customDialog = Dialog(this)
        customDialog.setContentView(R.layout.custom_dialog)
        val closeButton = customDialog.findViewById<Button>(R.id.closeButton)
        closeButton.setOnClickListener {
            customDialog.dismiss()
        }
        customDialog.show()
    }
}
```

4) How do activities, services, and the Android Manifest file work together to make an Android app? Can you describe their main roles and provide a basic example of how they cooperate to design a mobile app?

- When a user launches an Android app, the system starts the main activity. The main activity is responsible for creating the user interface and displaying the initial screen.
- If the app needs to perform a long running task, it can start a service. Services run in the background and do not have a user interface.

• Android Manifest file plays a role in all of this by declaring the components of the app and the permissions that they need. The system uses this information to determine how to start and manage the app's components.

Ex:

1. The user launches the app by tapping on its icon.
2. The Android system starts the main activity of the app.
3. The main activity displays the user interface and handles user input.
4. The user taps on a button in the main activity to start a service that downloads a file from the Internet.
5. The service is started and begins downloading the file.
6. The user continues to interact with the main activity while the file is downloading.
7. The service finishes downloading the file and sends a notification to the user.
8. The user taps on the notification to open the file.
9. The Android system starts an activity that can open the file.

5) How does the Android Manifest file impact the development of an Android application? Provide an example to demonstrate its significance.

Android Manifest plays a crucial role in the development of an android application as it serves as the blueprint that defines the app's structure. Here are some key ways in which the Android Manifest file impacts apps development:

1. Component Declaration: The manifest file is where you declare all the components of Android app. These declarations inform

the Android System about the app's entry points and functionalities.

2. Permissions: Declaring permissions in the manifest file is essential for ensuring that app functions correctly and securely.
3. Intent Filters: They enable app to handle implicit intents and respond to events like opening a specific file type.
4. Launch Configuration: It specify which activity should serve as the main entry point of app. This activity is launched when the app starts.
5. App Meladata: The manifest file contains meladata about the app, such as its name, icon, version and package name.

Ex:

```
<application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name">
    <activity
        android:name=".MainActivity
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name=".ChatActivity"
        android:label="Chat">
    </activity>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <receiver
```

```
android:name=".SmsReceiver">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
</application>
```

- MainActivity is declared as the launcher activity, which is the first screen.
- ChatActivity is another activity that allows users to chat with others
- Permissions for internet access and sending SMS messages are declared.
- An intent filter is set up for 'SmsReceiver' component to listen for incoming SMS messages.

6) What is the role of resources in Android development? Discuss the various types of resources and their significance in creating well-structured applications. Provide examples to clarify your points.

Resources play fundamental role in Android development by providing a structured way to manage assets, values, layouts and other elements used in app. They help create flexible, maintainable and device independent application. The various types of resources and their significance with examples:

1. Layout Resources:
- type: xml files in the 'res/layout' directory
- significance: Define the structure and appearance of the app's user interface.
- Ex: 'activity_main.xml' defines the layout of your main activity, specifying UI components like buttons, text views and their arrangement.

```
<Button
    android:id="@+id/myButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Submit" />
```

2. Drawable Resources:
   - type: Images and drawable assets in the 'res/drawable' directory
   - significance: Store graphics, icons, and images used in your app.
   - Ex: 'ic_launcher.png' is the app's launcher icon.

3. String Resources:
   - type: Strings defined in XML files under 'res/values'.
   - significance: Store text strings, making it easier to provide translations and maintain consistency.
   - Ex: res/values/strings.xml contains string resources.

```
<string name="app_name">My app</string>
<string name="welcome_message">Welcome</string>
```

4. Color Resources:
   - type: colors defined in XML files under 'res/values'
   - significance: store color values, ensuring consistency in the app's design.
   - Ex: res/values/colors.xml defines color resources.

```
<color name="primary_color">#007ACC</color>
<color name="accent_color">#FFA500</color>
```

5. Style Resources:
   - type: styles defined in XML files under 'res/values'
   - significance: define reusable styles for UI components.
   - Ex: res/values/styles.xml defines style

```
<style name="MyButtonStyle">
<item name="android:background">@drawable/my_button</item>
<item name="android:textColor">@color/primary_color</item>
</style>
```

6. Dimension Resources:
- type: dimensions defined in XML files under res/values
- significance: store dimension values, ensuring consistent layout
- Ex: res/values/dimens.xml defines dimension resources.
<dimen name="margin_large">16dp</dimen>
<dimen name="padding_medium">8dp</dimen>

7. Raw Resources:
- type: files stored in the res/raw directory
- significance: store non-xml files, such as JSON data, audio.
- Ex: store a JSON file for app configuration.

7) How does an Android service contribute to the functionality of a mobile application? Describe the process of developing an Android service.

Contributions of Android Services:

1. Background processing:- services allow apps to perform tasks in the background without blocking the user-interface.
2. Long running operations:- services are ideal for handling operations that require more time to complete.
3. Inter-component communication:- services enable components like activities, broadcast receivers and other services to communicate with each other efficiently.
4. Foreground services:- Android services can run in the foreground, even when the app isn't in the foreground. This is useful for features that require ongoing user interaction.

Process of developing an Android Service:

1. Define the service class:- Create a kotlin class that extends the service class. override methods like onCreate(), onDestroy(),

onStartCommand() to define the behavior of service.

2. Configure service in Manifest: Declare service in the Android Manifest.xml file to inform the Android system about its existence and configuration.
3. Start or bind the service: Decide whether you want to start a service or bind it to other components.
4. Implement service logic: in service class, implement the specific logic your service needs to perform its task.
5. Handle Lifecycle: Release resources when they're no longer needed.
6. Interact with other components: use appropriate mechanisms to facilitate the communication
7. Foreground services (optional): ~~Implement~~ if your service proper needs to run in the ~~foreground~~.
8. Testing: Thoroughly test your service to ensure it functions as expected.
9. Optimization: Optimize your service for performance and resource efficiency to minimize battery usage
10. Error handling and logging: implement proper error handling and logging mechanisms to diagnose and address issues that may occur while service is running.

05/10/23