Name: Chaudhary Khushi H. Enr. No: 210120110010
Class: CE-IT-B Batch: 5B4

Ganpat University | U.V. Patel College of Engineering

U.V. Patel College of Engineering

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

2CEIT5PE5: Mobile Application Development Assignment: 1

Q-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.

→ As per my knowledge, one notable trend in the mobile app industry that was influencing the Android Platform was the rise of Progressive web APPS (PWAs). PWAs are web applications that offer app-like experiences directly through web browsers.

- Impact on Android APP Developers:

1. Cross-Platform Compatibility: PWAs are designed to work seamlessly across various platforms and devices, including Android. Developers had to consider creating PWAs alongside traditional Android apps to ensure broad accessibility.
2. Enhanced User Experience: PWAs aimed to provide a smoother and more engaging user experience, which set higher expectations for Android app developers. This encouraged them to focus on improving the quality and performance of their apps to compete effectively.
3. Progressive Enhancement: Developers needed to adopt progressive enhancement strategies to ensure that Android apps remained competitive by offering progressive and responsive user experiences, similar to PWAs.

- Impact on Businesses in the Mobile APP Industry:

1. Cost Savings: Businesses could potentially save on development costs by investing in a single PWA that works across multiple platforms, including Android, rather than building separate native apps.
2. Increased Reach: PWAs enabled businesses to reach a wider audience, including users with Android devices, without relying solely on app store distribution. This broader reach could lead to increased user acquisition.

3. Improved Engagement: The focus on delivering app-like experiences through PWAs encouraged businesses to prioritize user engagement and retention, ultimately benefiting their mobile strategy.
4. Competition and Innovation: The rise of PWAs introduced competition, driving businesses to innovate their Android apps to keep up with evolving user expectations and technology trends.

2. What is the purpose of an Inflater of Layout in Android development and how does it fit into the architecture of Android Layouts?

→ In Android development, an "Inflater" refers to the LayoutInflater, which plays a crucial role in creating a user interface (UI) from XML layout files. Its primary purpose is to take an XML layout resource and convert it into a corresponding view object in memory. Here's how the LayoutInflater fits into the architecture of Android Layouts:

1. XML Layout files: In Android, UI components are often defined using XML layout files. These files describe the structure and appearance of the UI, specifying things like widgets, their properties and their placement within the UI.
2. Layout Inflation: When your Android app runs, it needs to turn these XML layout files into actual view objects that can be displayed on the screen. This process is known as "Layout Inflation".
3. Layout Inflater: It is responsible for reading the XML layout files and instantiating the corresponding view objects in memory. It takes the XML file as input, parses it, and creates the view objects, such as TextViews, Buttons, etc. as specified in the XML.
4. Dynamic UI Creation: Layout inflation is particularly valuable when you need to create UI elements dynamically, for example, in response to user interactions or when working with RecyclerViews to display lists of items.
5. Binding Data: Once the view objects are created, they can be further customized and data can be bound to them. This is typically done using data binding techniques or programmatically setting properties and values.
6. Displaying UI: After inflation and customization, the view objects can be added to the app's layout hierarchy and displayed on the screen.

Q-3 Explain the Concept of a Custom Dialog Box in Android applications. Provide examples to illustrate its use.

⇒ In Android applications, a custom Dialog Box is a pop-up window that overlays the current activity and is often used to interact with the user, gather Input or display Information. overview of a custom Dialog Box :

- Purpose : custom-dialogs are used when you want to present information, receive user Input or perform actions within a self-contained, isolated UI element that temporarily interrupts.
- Components : A custom dialog typically consists of various UI elements like buttons, textviews, images or input fields, tailored to the specific interaction you want to facilitate.
- Customization : Developers can design the dialog's appearance, layout, and behavior according to their app's branding or specific requirements. This customization allows for creativity in design and functionality.

– Simple example of Creating and using a custom dialog in Android.

```
fun showCustomDialog() {
    val customDialog = Dialog(this)
    customDialog.setContentView(R.layout.custom_dialog)
    val messageTextview = customDialog.findViewById<TextView>
                                (R.id.messageTextview)
    val okButton = customDialog.findViewById<Button>(R.id.
                                okButton)

    messageTextview.text = "This is a custom dialog!"
    okButton.setOnClickListener {
        customDialog.dismiss()
    }
    customDialog.show()
}
```

⇒ use cases of custom dialog box : • login • conformation Dialog • settings • Informational Pop-up • Media Playback Controls

Q-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?

→ 1. Activities:
- Role: Activities represent individual screens or UI components in an Android app. They manage the user interface and user interactions.

2. Services:
- Role: Services are background components that perform long-running operations or handle tasks that don't require a user interface. They can run even if the app's UI not visible.

3. Android Manifest file:
- Role: The AndroidManifest.xml file is like the app's blueprint. It declares the app's components and defines how they interact with the Android system and other components.

Example: In AndroidManifest.xml, you specify which activities are part of your app, their Launch modes, Permissions and service declarations. This file acts as a blueprint for Android System to understand your APP's structure and behavior.

→
```
class MainActivity : APPCompatActivity() {
    override fun onCreate(Saved InstanceState : Bundle?) {
        Super.onCreate(Saved InstanceState)
        SetContentview(R.layout.activity_main)
        StartserviceButton.SetonclickListener {
            val serviceIntent = Intent(this, Notificationservice::class.java)
            Startservice(serviceIntent) } } }
```

→
```
class Notificationservice : IntentService("Notificationservice") {
    override fun onHandleIntent(Intent: Intent?) {
        if (Intent != null) {
            createNotification() } }
Private fun createNotification() {
    val channelID = "my_channel"
    if (Build.version.SDK_Int >= Build.version_codes.O) {
        val name = "my channel"
        val notificationManager = getSystemservice(NotificationManager::
                                                    class.java)
        notification Manager.create Notification channel (channel) }
    val Builder = NotificationCompat.Builder(this, channelID)
        .SetSmallIcon(R.drawable.ic_launcher_foreground)
        .Set Content Text("This is notification from service.")
} }
```

Q5 How does the Android Manifest file impact the development of an Android application? Provide an example to demonstrate its significance.

→ The Android Manifest file is a crucial component in the development of an Android application. It serves several important purposes, and its content significantly impacts how the android system interacts with and manages your APP.

Significance of the Android Manifest File:

- App Configuration
- Component Declaration
- Permissions
- Intent Filters
- App Lifecycle

Example:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity">
            .... Declare additional activities here ....
        </activity>
        <uses-permission android:name="android.permission.INTERNET" />
```

... Declare required permissions here ....
</application>
</manifest>

Q-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 a fundamental role in Android development by providing a structured way to manage assets, values, layouts and other elements used in your 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.
- Example: '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 = "Click me" />
```

2. Drawable Resources:
- type: Images and drawable assets in the 'res/drawable' directory.
- significance: Store graphics, icons, and images used in your app.
- Example: '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.
- Example: 'res/values/strings.xml' contains string resources.

```
<String name = "app_name"> MY APP </String>
<String name = "welcome_message"> welcome to my APP </String>
```

4. Color Resources:
- type: Colors defined in XML files under 'res/values'.
- Significance: Store color values, ensuring consistency in the app's design.

Example: '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.
- Example: '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 a consistent layout
- Example: '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.
- Example: Store a JSON file for app configuration.

Q-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, such as playing music.

3. Inter-component communication: Services enable components like activities, broadcast receivers and other services to communicate with eachother 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, like music playback.

Process of Developing an Android service:

1. Define the service class: create a new Java or Kotlin class that extends the 'Service' class. - override methods like onCreate(), onDestroy(), onStartCommand() to define the behavior of your service.

2. Configure Service in Manifest: Declare your service in the Android Manifest.xml file to inform the Android System about its existence and configuration. <service android:name=".MyService" />

3. Start or Bind the service: Decide whether you want to start the service or bind it to other components. use StartService() or bindService().

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 and consider using 'stopself()' or 'stopService()'.

6. Interact with other components: use appropriate mechanisms like intents, broadcasts or callbacks to facilitate communication

7. Foreground services (optional): If your service needs to run in the foreground, 'startforeground()'.

8. Testing: Thoroughly test your service to ensure it functions as expected, including handling various scenarios like network failures

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.