Suthar Ruhul N. 210120111148

Ganpat University | U.V. Patel College of Engineering

MEHSANA DISTRICT EDUCATION FOUNDATION SANCHALIT
U. V. Patel College of Engineering
GANPAT UNIVERSITY, KHERVA - 384 012 DIST. MEHSANA. (N.G.)

Assignment-1 (MAD)

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 business in the mobile app industry

Ans→ One significant trend in the Android app industry is the increasing emphasis on user privacy and data security

→ Impact on Android App Developers.

1. Enhanced permissions and consent:
→ Developers had to be more transparent about the data their apps collect and request explicit user content. This meant redesigning permission dialogs and ensuring that users understood why certain data was being collected

2. Limitations on Advertising
→ For apps relying on advertising revenue, changes in ad tracking and targeting due to privacy concerns affected their monetization strategies. Developers needed to adapt to these changes, possibly exploring alternative monetization models.

3. Data Handling and Storage:
→ Developers had to review how they handled and stored user data, implementing stricter data protection measures. This could lead to increased development time & costs.

→ Impacts on Businesses.

1. Compliance Costs:

⇒ Businesses operating in the android app industry needed to allocate resources for compliance with stricter data privacy regulations. This could include ~~global~~ legal and technical measures to ensure data protection.

2) Monetization challanges

→ Businesses relying heavily on user data for advertising and personalized content faced challanges in maintaining their revenue streams. They needed to find new ways to engage users and generate income.

3. Reputation management:

→ Privacy breaches or mishandling of user data could result in severe reputational damage. Building and maintaining trust with users become even more critical.

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

→ In android app development, think of the "Inflater" like a magic tool. it helps turn your design plans into actual buttons, text boxes, and other things you see on your phones. screen.

→ Purpose of LayoutInflator.

1) Dynamic UI Inflation: LayoutInflater is used to create instances ~~insto~~ of Android view objects from xml Layout resource files at runtime.

2) Reusability: It promotes the reusability of UI components by defining their structure and appearance in xml Layout files, making it easier to instantiate and populate them in different parts of an App.

3) Separation of concerns: layoutInflater helps maintain clear separation b/w the UI design and the code that manipulates and interacts with these UI elements.

→ Architecture of Android Layouts

1) xml Layout files: Developers design the layout & structure of UI elements in xml layout resource file.

2. Activity / Fragment: In the Java or Kotlin code of an android activity or fragment, developers use the LayoutInflater to "inflate" or parse the xml layout files, creating a hierarchy of view objects. This is typically done within the 'onCreate' method.

3. View Hierarchy: The result of inflating the layout xml is a hierarchy of view objects, with the root view being the top-level layout.

4. Data Binding & Event Handling: Developers often bind data to these views using data binding libraries or handle user interactions by attaching event listeners.

5. Rendering on the screen: The android system is responsible for rendering this hierarchy of views on the device screen according to the layout specifications defined in the xml file.

Q-3 Explain the concept of customDialogBox in Android application. provide examples to illustrate its use.

→ A custom Dialog Box in Android applications is a pop-up window that developers can design and customize to show specific information, receive input from users or perform actions without navigating to a new screen or activity. custom Dialog Boxes are helpful for displaying messages, alerts forms, or any custom content in controlled and visually appealing manner.

1. Design Flexibility: custom DialogBoxes allowes developers to create unique and tailored user interface.

2) Contextual Use: They are typically used when you want to capture user input or show information without taking the user to a different screen.

3) User Interaction: Custom Dialog Boxes can contain buttons, text-fields, checkboxes & any other UI element, allowing users to interact with the content inside the dialog.

→ Examples of custom Dialog Box User

1) Confirmation Dialog: A common use case is asking the user for confirmation before performing a critical action

2) Login or Registration Dialog: Instead of navigating to a separate screen for login or registration, a custom dialog box can pop up, prompting the user to enter their credentials.

3) Error messages: When there is an error, such as network issues or invalid input, a custom dialog can display an error message with details, helping the user understand and correct the problem

4) Date & time picker: You can create a custom dialog box for selecting dates or times, providing a more user-friendly way to input this information.

Code

```
import android.app.AlertDialog
import android.appcontent.DialogInterface
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

class YourActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val builder = AlertDialog.Builder(this)
        builder.setTitle("Custom Dialog example")
        builder.setMessage("This is a custom dialog box ex")
        builder.setPositiveButton("OK") { dialog, which -> }

        val dialog = builder.create()
        dialog.show()
    }
}
```

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 how they cooperate to design a mobile app?

→ Activities, services, & the Android manifest file are essential components in the android app architecture each with distinct role that contribute to the functionality and behaviour of an app.

1. Activities.

→ Role: Activities represent the user interface and screen of an Android app. they handle user interactions, display UI elements, and manage the UI flow.

→ Example: Imagine a simple note-taking app. Each screen of the app, such as the note list, note editing, and settings, can be implemented as separate activities.

2. Services

→ Role: Services run in the background and perform long-running or background tasks without a user Interface

→ Example: In our note-taking app, you might have a service that periodically backs up notes to a cloud server without showing a user Interface

3. Android manifest file:

→ Role: The Android manifest file is a configuration file

provides essential information about the app to the Android system. it declares the app's components, permission, and other settings.

→ Example: In the manifest file, you define which activities are part of your app, specify permissions & declare services your app uses.

→ How they cooperate:-

1) Activities
   → The app starts with an activity showing a list of notes.
   → When the user taps on a note, another activity opens to display and edit the note's content.
   → Users can navigate between activities using buttons or gestures.

2) Services
   → While the user is using the app, a service runs in the background to periodically save the user's notes to cloud storage.
   → This service doesn't have a user interface but operates independently to ensure data is continuously backed up.

3) Android manifest file
   → In the manifest file, you declare the activities and services used in your app.
   → You specify permission like "INTERNET" to allow the app to access the internet for cloud backups.
   → The manifest file also defines which activity to start when the app launches.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/
    package="com.example.mynotesapp">
    <application>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
```

Q-5) How does the Android manifest file impact the development of an android application? provides an example to demonstrate its significance.

→ The android manifest file impacts app development by:

1. Component Declaration: Declaring app components to define the app's structure.
   ex <activity android:name=".mainActivity" />
2. App permissions: specifying permissions for accessing device resources.
   ex <uses-permission android:name="android.permission.CAMERA" />
3. Intent filters: Defining how the app responds to external actions or requests.
   ex: Registering to open PDF files when tapped.

```
<activity android:name=".PdfViewerActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="application/pdf" />
    </intent-filter>
</activity>
```

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

→ Resources in Android development are essential components that helps you create well-structured and flexible applications. they serves several purposes, such as seperating code from content, adapting to different devices, and simplifying localization. Here are the main types of resources and their significance.

1.) Layout Resources

- xml Layouts: These define the structure and appearance of your apps user interface. They help keep the UI separate from code logic, making it easier to maintain & adapt
- Example : A layout xml file specifies how elements like buttons and text fields are arranged on the screen.

2) Drawable Resources

- images and icons : Drawable resources store images, icons, and other graphics used in your app. Diff. versions can be provided for diff. screen densities

- example : You might have 'ic_launcher.png' for the app icon & separate versions for low, medium & high-density screens.

3) String Resources:

- Text and strings : storing text in resource files allows for easy localization and updates without modifying code.

- Example : A string resource ~~contains~~('app_name') contains the app's name, which can be changed for different languages.

4) Color Resources

- colors : By defining colors in resources, you can maintain a consistent color scheme across your app and easily switch themes.

- Example : A color resource (primary_color) define the primary

colors used in the app's UI elements

5) Style Resources.

- Themes and Styles : Styles define the appearance of UI elements, making it simple to apply consistent styling across app.

- Examples : You can create a custom style (AppTheme) to define fonts, colors and other visual attributes

6) Dimension Resource

- Sizes and Dimensions : Storing sizes and margins in res file makes it easy to adjust layouts for different screen sizes and orientations

- Example : A dimension resource (margin_small) defines a consistent margin size for elements.

7) Raw Resources

- Raw Data : You can store non-compiled resources like audio, video, or text files in the 'res/raw' directory.

- Example : Storing a JSON file in the 'raw' folder for configuration data.

8) Animations : and Drawable Animation Resources.

Animation : You can define animations in xml resource file, making it simple to

reuse and apply animations to UI elements.

Example: A resource file ('fade_in.xml') can define a fade in Animations for an ImageView.

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

→ An Android service plays a crucial role in the functionality of a mobile application by allowing tasks to run background, even when the app is not actively in use.

- Contribution of Android service:

1) Background processing: Services run tasks in the background ensuring the essential functions music playback, location tracking or data syncing continue without disrupting the user's interface.

2) long-Running operations: Services are ideal for operations that take a long time to complete such as downloading large files or performing complex calculations, without causing the app to freeze.

3) foreground Services: Some services can run in foreground displaying a persistent notification to keep the user aware of ongoing tasks like navigation or chat application.

4) Inter-component Communication.

→ Services can communicate with other app components (activities, fragments) through interfaces allowing data exchange and coordination.

→ Developing an Android service:

1) Create a service class:
→ Extend the 'Service' class or one of its subclasses like 'IntentService' or 'JobService'.
→ Implement the service's functionality within the 'onCreate' & 'onStartCommand' methods.

2) Declare in the manifest
→ Register your service in the AndroidManifest.xml file to make it accessible to the system and other components.

3) Service Lifecycle:
→ Understand the service's lifecycle methods [onCreate, onStartCommand, onBind, onDestroy] and override them as needed.
→ Service can run in three methods: foreground, background, or bound. Choose the appropriate mode based on your app's requirement

4) start and stop the service
→ Start a service using 'startService(Intent)' or bind to it using 'bindService(Intent, ServiceConnection, int)'
→ Stop a service when it's no longer needed using 'stopService(Intent)' or 'stopSelf()'.

5) Foreground Services
→ To create a foreground service, provide a notification that informs the user about ongoing tasks.
→ Use 'startForeground()' to start a service in the foreground mode.

6) Thread management.
→ when performing time-consuming operations, consider using worker threads or AsyncTask to prevent blocking the main UI thread.

7) Communications
→ Use Intent extras, broadcast receivers, or interfaces to enable communication b/w Services and other app components.

8) Cleanup and Resource management.
→ Ensure that you release resources and stop the service when it is no longer needed to prevent unnecessary battery drain.

9) Testing
→ Thoroughly test your service to ensure it works as expected, including scenarios like app backgrounding, task interruptions, and restarts.

Dads
6-10-23