17/10/23

ASSIGNMENT 3

Unit-5&6 Android services & Multimedia and Networking in Android.

Q.1 What is Android Services ? Explain life cycle of services.

—> Android services are a fundamental component that allows you to perform background tasks or run long-running processes without requiring a user interface. Services are often used for tasks like playing music in the background, downloading files.

—> The life cycle of Android services :-

i) Not Started : This is the initial state where the service is not yet started and it stays like that until some component (eg: Activity or another service) starts it using 'startService()' or 'bindService()'.

ii) Started : When a service starts using 'startService()' it enters the "Started" state. The service can continue to run even if the component that started it is destroyed or goes into the background.

iii) Bound : When a service is bound to a component using 'bindService()', it enters the 'Bound' state, where the component can interact with service by making method calls on the service's interface.

iv) Running : This state is a combination of being

started and bound. It's in this state when its both started using 'startService()' and bound using 'bindService()'.

v) Destroyed: When a service is no longer needed, it can be stopped using 'stopService()' or 'unbindService()'. Once all components that have started, release their connection, the service enters the "Destroyed" state and is terminated.

Q.2 — Explain Management of SMS in Android.

Managing SMS (Short Message Service) involves sending, receiving and handling text messages within your application. Some key components and steps to manage SMS:

i) Permissions: To send and receive SMS, your application must request the necessary permissions in the AndroidManifest.XML file and request them at runtime if you are trageting Android 6.0 (API level 23) or higher. The relevant permissions are:

> 'SEND_SMS': Send SMS messages.
> 'RECEIVE_SMS': Receive and read SMS messages
> 'READ_SMS': Read SMS messages.
> 'RECEIVE_MMS': Receive and read MMS messages.

ii) Broadcast Receiver: To receive incoming SMS messages, you can register a 'BroadcastReceiver'

that listens for the 'android.provider.Telephony.SMS_RECEIVED' intent. When a new SMS arrives, your app's 'BroadcastReceiver' is triggered, and you can extract and process the message.

iii) Sending SMS: To send an SMS, you can use the 'SmsManager' class.

eg:
```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage(phoneNumber, null,
                           message, null, null);
```

iv) Processing SMS: When your 'BroadcastReceiver' is triggered, you can access the incoming SMS message's details and perform actions on it.

v) SMS Content Provider: If you need to access stored SMS messages, you can use the SMS content provider. You can query the 'Telephony.Sms' content provider to receive SMS messages stored on the device.

eg:
```
Uri uri = Uri.parse("content://sms/inbox");
Cursor cursor = getContentResolver().query(uri, null,
                null, null, null);
```

Q.3 Explain Notification Service in Android.

A Notification Service refers to a system component that allows applications to display notifications to the user. Notifications are a way of for apps to provide information, updates

or alerts to users, even when the app is not actively running in the foreground.

i) Notification Manager : Notification Service is primarily managed by the 'NotificationManager' system service. It is responsible for creating, displaying and managing notification on the device.

ii) Notification Components : To create and display notifications, apps typically use three primary components :

① Notification Builder : This is an API that allows ~~you~~ developers to construct notification objects which enables you to specify various properties of the notification, including the title, content, icon, actions and more.

② Notification Channel : This allows you to group notifications and give users more control over how they receive notifications from your app. You can define different ~~that~~ channels for different types of notifications and let users customize their behaviour.

③ Pending Intent : It is a way to ~~are~~ create an intent that can be executed at a later time, often in response to a user's actions on the notification which allows you to define what

should happen when the user interacts with the notification.

eg: To create and post a simple notification

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My Notification")
    .setContentText("This is a notification")
    .setContentIntent(pendingIntent);
NotificationManager notificationManager = (NotificationManager)
    context.getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFICATION_ID, builder.build());
```

Q.4 Explain Alarm Service in Android.

—4 Alarm Service is a system service that allows developers to schedule and execute code at a specified time, even when the app is not actively running. They are commonly used to perform tasks in the background such as data synchronization, notification and other periodic operations. There are two main types of alarm:

① One-Time Alarms: These are alarms that trigger a specific action at a specified time in the future. For example, ~~seche~~ scheduling a daily reminder to take medication.

② Repeating Alarms: These alarms execute a specific action repeatedly at a fixed interval. For example, repeating alarm for hourly updates from server.

- Here's how Alarm Service work :

i) Alarm Manager

The Android Alarm Service is primarily managed by the 'AlarmManager' class. Used to schedule one-time or repeating alarms.

ii) Pending Intent

To specify the action that should be triggered when the alarm fires, developers use a 'PendingIntent', which is a way of to encapsulate an intent and execute it at a later time.

iii) Scheduling Alarm

To schedule an alarm, you create a 'Pending Intent' that represents the action you want to execute, set the Alarm Type and time and then use 'AlarmManager' to set the alarm.

eg: To schedule a one-time alarm to trigger an action

```
AlarmManager alarmManager = (AlarmManager)
        getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(this, YourReceiver.class);
PendingIntent pendingIntent = PendingIntent.
        getBroadcast(this, 0, intent, 0);
long alarmTime = System.currentTimeMillis() + 10000;
alarmManager.set(AlarmManager.RTC, alarmTime,
        pendingIntent);
```

eg: Repeating Alarms

```
alarmManager.setRepeating(AlarmManager.RTC_WAKEUP,
        initialTime, interval, pendingIntent);
```

Q.5 — 4 Explain Sending an Email in Android.

Sending an email in an Android app involves using the Android Intent system to launch an email application on the device, prepopulate the email's subject, recipient, body and other fields.

i) Permissions : To send an email, your app needs the 'android.permission.INTERNET' permission in your AndroidManifest.xml file for opening the email app. The requires when your app targets Android 6.0 (API level 23) or higher.

ii) Create an Intent : You can create an 'Intent' with the 'ACTION_SENDTO' action, which is used to send data to a specified ~~recep~~ recipient through a mail application.

iii) Specify the Email Address : To specify the recipient's email address, use the 'setData' method with a 'Uri' containing the 'mailto:' scheme.

iv) Set Subject and Body : You can set the email's subject and body using the 'EXTRA_SUBJECT' and 'EXTRA_TEXT' fields in the 'Intent'.

v) Launch the Email App : To open the email app with the prepapulated fields, you use the 'StartActivity' method with the intent.

eg:
```
Intent emailIntent = new Intent(Intent.ACTION_SENDTO);
Uri uri = Uri.parse("mailto:recipient@example.com");
emailIntent.setData(uri);
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email Subject");
emailIntent.putExtra(Intent.EXTRA_TEXT, "This is the email body.");
try {
    startActivity(emailIntent);
} catch (ActivityNotFoundException e) {
    // Handle the case where no email app is
    // installed on the device.
}
```

Q.6 Explain how to handle maps and location data services.

→ Handling maps and location data services involves utilizing the Android location services and integrating mapping functionality, often using the Google Maps API.

i) Acquire Permissions: To access location services, you need to request the necessary permissions in the AndroidManifest.xml file and request them at runtime if you are ~~the~~ targeting Android 6.0 (API level 23) or higher. The relevant permissions are: 'ACCESS_FINE_LOCATION' (for precise location information) and 'ACCESS_COARSE_LOCATION' (for less precise, network-based location information).

ii) Request Location Updates : To obtain the device's current location, you can use the 'Location Manager' or the more modern 'FusedLocation ProviderClient' from Google Play Services. This allows you to request periodic location updates

iii) Display Maps : To display maps, you can integrate Google Maps through the Google Maps Android API. Add the 'MapView' to your layout to display the map.

eg:
```
<com.google.android.gms.maps.MapView
    android:id="@+id/mapView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

iv) Initialize Maps : Initialize the map using the API key and the 'MapView' or 'MapFragment'.

eg:
```
MapsInitializer.initialize(context);
mapView.onCreate(savedInstanceState);
```

v) Customize Maps : You can customize it by adding markers, polygons, overlays.

eg:
```
// Add a marker to the map
LatLng location = new LatLng(latitude, longitude);
googleMap.addMarker(new MarkerOptions().position
        (location).title("Marker Title"));
```

vi) Handle Map Events : You can listen (expect) for various map events, such as marker clips or map clicks

to perform actions based on user interactions.

eg: googleMap.setOnMapClickListener(LatLng -> {
// Handle Map click
});

Q.7 a) Enlist and Explain Android Web Services.

—> i) RESTful Web Services

REST (Representational State Transfer) is a popular architectural style for building web services. Android apps can use HTTP methods like GET, POST, PUT and DELETE to send and receive data in JSON or XML format.

ii) SOAP Web Services

SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information using XML. While not as common as REST, apps can interact with SOAP web services.

iii) WebSocket Services

WebSocket is a protocol that enables two-way real-time communication between a client and server. Apps can use its libraries to establish and maintain persistent connections for real-time updates.

iv) GraphQL

GraphQL is a query language for APIs, allowing clients to request only the data they

need. Apps can communicate with GraphQL APIs to retrieve data efficiently.

v) JSON-RPC and XML-RPC:

These are Remote Procedure Call (RPC) protocols that enable the invocation of methods and functions on a remote server. JSON-RPC uses JSON for data encoding and XML-RPC uses XML.

b) Explain AndroidX Library.

→ AndroidX is a major improvement and evolution of the original Android Support library. It provides android app developers with a set of libraries and tools that are backward compatible and designed to work with modern Android development practices. It simplifies the development process and eliminates many of the conflicts that developers experienced with the Older Support Library.

Key features of AndroidX include:

- Backward Compatibility
- Modular Architecture
- Package Refactoring
- Improved Version Management
- New components and Features

Q.8 Explain JSON object and passing.

→ A JSON (Javascript Object Notation) object is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON is widely used for data exchange between a server and a client and it has become the standard format for web APIs. JSON data is represented as key-value pairs.

- JSON Object Structure:

A JSON object is enclosed in curly braces '{}' and contains a set of key-value pairs. Keys are strings which are associated with a value which can be string, number, boolean, null, array or nested JSON objects.

eg:
```
{
    "name": "Tony",
    "age": 25,
    "isStudent": false,
    "address": {
        "street": "123 Main St",
        "city": "Smallville"
    }
}
```

- Passing JSON objects:

JSON Objects can be passed in various contexts, including:

ii) HTTP Request and Response : While interacting with web APIs, you often send JSON objects as part of an HTTP request and receive as part of an HTTP response.

iii) Local Data Storage : You can save JSON objects in local storage such as Shared Preferences or SQLite database, to persist app data between sessions.

iii) Remote Messaging : JSON objects are often used in push notifications or cloud messaging to deliver structured data to mobile devices.

Q.9 How Android web services works ? & Discuss with code snippet

→ Android web services allows the app to communicate with remote servers or other web-~~servi~~ based resources over the internet. They are crucial for fetching data, sending data and interacting with online services. ~~Or~~ One common way to interact with web services is by making HTTP requests.

eg:
```
// import necessary packages.
// Now inside the predefined code
    dataTextView = findViewById(R.id.dataTextView);
    new FetchDataFromWebService().execute();
// After that block of code.
```

```
private class FetchDataFromWebService extends
                AsyncTask<void, void, String> {
    protected String Background(void... voids) {
      try {
        URL url = new URL("LINK");
        HttpURLConnection connection = (HttpURLConnection)
                url.openConnection();
        connection.setRequestMethod("GET");
        int responseCode = connection.getResponseCode();
        if (responseCode == HttpURLConnection.HTTP_OK) {
          BufferedReader reader = new BufferedReader
            (new InputStreamReader(connection.getInputStream()));
          StringBuilder response = new StringBuilder();
          String Line;
          while ((Line = reader.readLine()) != null) {
            response.append(Line);
          }
          reader.close();
          return response.toString();
        } else {
          return "Error: " + responseCode;
        }
      } catch (IOException e) {
        e.printStackTrace();
        return "Error: " + e.getMessage();
      }
    }
    protected void onPostExecute(String result) {
      super.onPostExecute(result);
      dataTextView.addText(result);
    }
}
```

Q.10 Explain multimedia recording in android.

—4 Multimedia recording refers to the process of capturing various types of media like audio, video and images using a mobile's built-in hardware like camera, microphone and sensors.

i) Audio Recording: Android allows you to record audio using the 'MediaRecoder' class. To capture audio, you need to set the audio source (setAudioSource), format (setOutputFormat) and output file (setOutputFile) and then start and stop the recording process.

ii) Video Recording: Android ~~allows~~ provides the 'Media Reader' class for this as well. You need to set the video source, format and output file and other parameters to start and stop video recording. (setVideoSource, setOutputFormat).

iii) Camera2 API: For more advanced video and image capture, you can use this API which provides greater control over camera settings and allows for custom images and video capture.

winter 02/11/23

iv) Handling Permissions: To access camera, storage or microphone, apps needs to request the appropriate permissions for Android 6.0 or higher.

v) Previewing Media: You can use the 'SurfaceView' or 'TextureView' to display camera preview.

~~~x~~~x~~~
~~~