UNIT V: WORKING WITH INTERNET, DATABASES AND PUBLISHING APPS

6.1 Shared preferences.

6.2 Downloading and Parsing Internet Resources,
Using the Download Manager.

6.3 Files access.

6.4 Introducing Android Databases, Introducing SQLite,
Content Values and Cursors,
Working with SQLite Databases.

6.5 Preparing for publishing.

6.6 Publishing to the Android Market.

6.1 Shared preferences

Shared Preference in Android are used to save data based on key-value pair. If we go deep into understanding of word: shared means to distribute data within and preference means something important or preferable, so Shared Preferences data is shared and preferred data.

Shared Preference can be used to save primitive data type: string, long, int, float and Boolean.
There are two different ways to save data in Android through Shared Preferences – One is using

1. Activity based preferences
2. Custom preferences.

1. Activity Preferences:

- For activity preferences developer have to call function **getPreferences (int mode)** available in Activity class
- Use only when one preference file is needed in Activity
- It doesn't require name as it will be the only preference file for this activity
- Developer doesn't usually prefer using this even if they need only one preference file in Activity. They prefer using custom getSharedPreferences(String name,int mode).

2. Custom Preferences:

- Developer needs to use **getSharedPreferences(String name,int mode)** for custom preferences
- Used in cases when more than one preference file required in Activity
- Name of the preference file is passed in first parameter

Mode And Its Type In Shared Preference:

There are three types of Mode in Shared Preference:

1. **Context.MODE_PRIVATE – default value (Not accessible outside of your application)**
2. **Context.MODE_WORLD_READABLE – readable to other apps**
3. **Context.MODE_WORLD_WRITEABLE – read/write to other apps**
4. **MODE_PRIVATE –** It is a default mode. MODE_PRIVATE means that when any preference file is created with private mode then it will not be accessible outside of your application. This is the most common mode which is used.
5. **MODE_WORLD_READABLE –** If developer creates a shared preference file using mode world readable then it can be read by anyone who knows it's name, so any

other outside application can easily read data of your app. This mode is very rarely used in App.

6. **MODE_WORLD_WRITEABLE –** It's similar to mode world readable but with both kind of accesses i.e read and write. This mode is never used in App by Developer.

```
SharedPreferences sharedPreferences =
context.getSharedPreferences("LoginDetails", Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString("Email", email);
    editor.putString("Password", password);
    editor.commit();
```

6.2 Downloading and Parsing Internet Resources, Using the Download Manager.

Step 1: Create a New Project

To create a new project in Android Studio

Step 2: Grant internet permission in the AndroidManifest.xml file

```
<uses-permission android:name="android.permission.INTERNET" />
```

Step 3: Working with the activity_main.xml file

Navigate to the app > res > layout > activity_main.xml and add the below code to that file. Below is the code for the activity_main.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:layout_gravity="center"
  android:gravity="center"
  android:orientation="vertical"
```

```
    tools:context=".MainActivity">

    <Button
        android:id="@+id/download"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Download Content" />

</LinearLayout>
```

Step 4: Working with the MainActivity.java file

Go to the MainActivity.java file and refer to the following code. Below is the code for the MainActivity.java file.

```
public class MainActivity extends AppCompatActivity {

    Button button;
    DownloadManager manager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = findViewById(R.id.download);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                manager = (DownloadManager)
getSystemService(Context.DOWNLOAD_SERVICE);
                Uri uri =
Uri.parse("https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf
/dummy.pdf");
                DownloadManager.Request request = new
DownloadManager.Request(uri);

request.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIB
LE);
                long reference = manager.enqueue(request);
            }
        });
    }
}
```

6.3 Files access

Android Operating System offers several storage methods to save your app data and images, audio files, docs, etc.

App-specific storage: It only stores the data that is used by your App.
Shared Storage: It stores files including images and documents.
Databases: It provides a local Database for your app (for example, SQLite)

To read data from the EXTERNAL ST0RAGE we have to add the following code to our AndroidManifest.xml file.

```
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

If you want to write/save some data to your device storage then you have to add the below permission to your AndroidManifest.xml file.

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Step 1: Create a New Project in Android Studio

Step 2: Add Permission to the AndroidManifest.xml file

Step 3: Add buildFeatures to build.gradle (Module:app)
Since in this project we used ViewBinding so we have to set ViewBinding=True.

```
buildFeatures {
    viewBinding = true
}
```

Step 4: Working with activity_main.xml

```
<ImageView
      android:id="@+id/background"
      android:layout_width="match_parent"
      android:layout_height="200dp"
      android:layout_marginTop="100dp"
      android:background="@color/white"
```

```
      app:layout_constraintBottom_toTopOf="@id/btn_access"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent"
      tools:ignore="MissingConstraints" />

   <!-- Button on which we apply OnClickListener to opening the gallery -->
   <Button
      android:id="@+id/btn_access"
      android:layout_width="200dp"
      android:layout_height="50dp"
      android:text="Access Images"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toBottomOf="@id/background" />
```

Step 6: Working with the MainActivity File

```
public class MainActivity extends AppCompatActivity {
  //ActivityResultLauncher to open the gallery
  private ActivityResultLauncher<Intent> openGallery =
registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(), result -> {
    //Check if result is OK and data is not null
    if (result.getResultCode() == RESULT_OK && result.getData() != null) {
      binding.getBackground().setImageURI(result.getData().getData());
    }
  });

  //ActivityResultLauncher to request permission
  private ActivityResultLauncher<String[]> requestPermission =
registerForActivityResult(new
ActivityResultContracts.RequestMultiplePermissions(), permissions -> {
    //Iterate through the permissions to check if they are granted
    for (Map.Entry<String, Boolean> entry : permissions.entrySet()) {
      String permissionName = entry.getKey();
      boolean isGranted = entry.getValue();
      if (isGranted) {
        Toast.makeText(this, "Permission is granted for accessing gallery",
Toast.LENGTH_LONG).show();
```

```
            Intent intent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
            openGallery.launch(intent);
        } else {
            if
(permissionName.equals(android.Manifest.permission.READ_EXTERNAL_STO
RAGE)) {
                Toast.makeText(this, "You denied the permission",
Toast.LENGTH_LONG).show();
            }
        }
    }
  });

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //Inflate the activity's layout
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());
    //Set onClickListener for button to request storage permission
    binding.getBtnAccess().setOnClickListener(v ->
requestStoragePermission());
  }

  //Show rationale dialog if permission is denied
  private void showRationaleDialog(String title, String message) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle(title).setMessage(message).setPositiveButton("Cancel",
(dialog, which) -> dialog.dismiss());
    builder.create().show();
  }

  //Request storage permission
  private void requestStoragePermission() {
    if (ActivityCompat.shouldShowRequestPermissionRationale(this,
android.Manifest.permission.READ_EXTERNAL_STORAGE)) {
      showRationaleDialog("Drawing App", "Drawing app needs to access your
external storage");
    } else {
      requestPermission.launch(new
String[]{android.Manifest.permission.READ_EXTERNAL_STORAGE});
    } } }
```

6.4 Introducing Android Databases, Introducing SQLite, Content Values and Cursors, Working with SQLite Databases.

SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

Database - Package

The main package is android.database.sqlite that contains the classes to manage your own databases

Database - Creation

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);
```

Database - Insertion

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class.

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS user(Username VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT INTO user VALUES('admin','admin');");
```

Database - Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatbase.rawQuery("Select * from user",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

There are other functions available in the Cursor class that allows us to effectively retrieve the data.

1 getColumnCount()

This method return the total number of columns of the table.

2 getColumnIndex(String columnName)

This method returns the index number of a column by specifying the name of the column

3 getColumnName(int columnIndex)

This method returns the name of the column by specifying the index of the column

4 getColumnNames()

This method returns the array of all the column names of the table.

5 getCount()

This method returns the total number of rows in the cursor

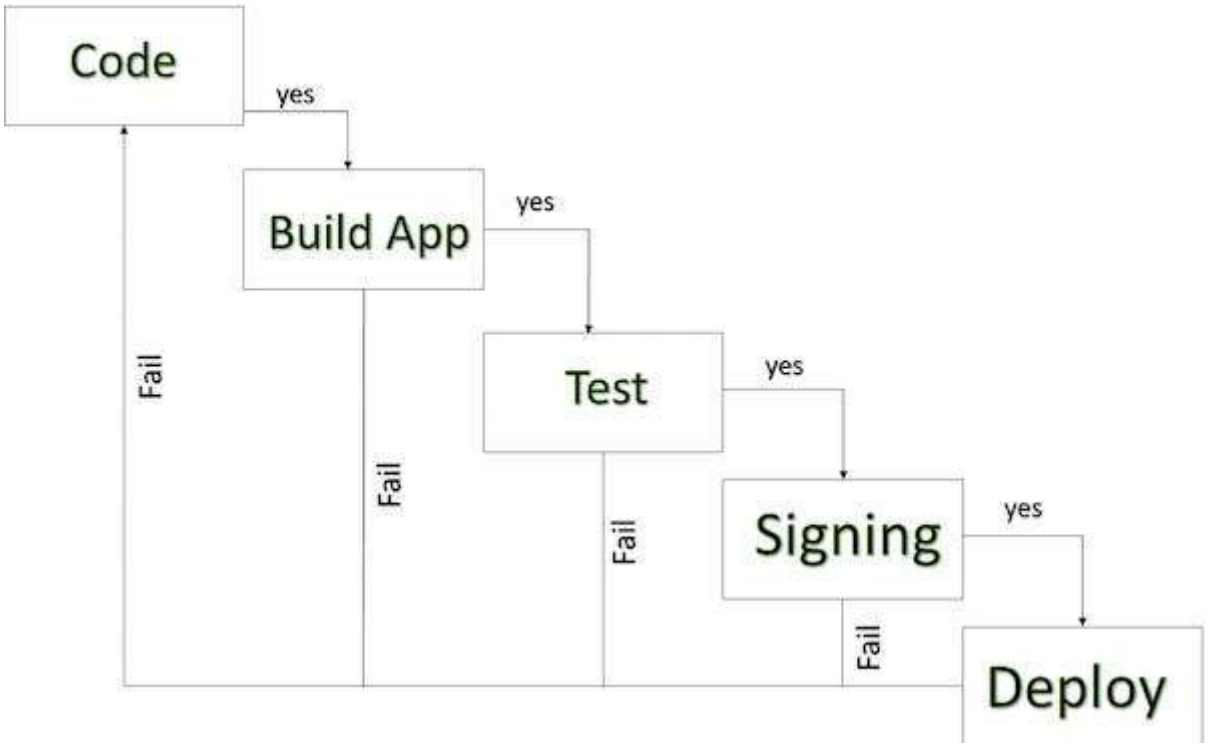
6 getPosition()

This method returns the current position of the cursor in the table

6.5 Preparing for publishing

Android application publishing is a process that makes your Android applications available to users.

publishing is the last phase of the Android application development process.



Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play

You can also release your applications by sending them directly to users or by letting users download them from your own website.

Regression Testing Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets.

Application Rating When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity.

Application Size Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices.

SDK and Screen Compatibility It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target.

Application Pricing Deciding whether you app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies.

Build and Upload release-ready APK The release-ready APK is what you you will upload to the Developer Console and distribute to users.

6.6 Publishing to the Android Market

The most important step is to register with Google Play using Google Play Marketplace. You can use your existing google ID if you have any otherwise you can create a new Google ID and then register with the marketplace.

You can use Continue to payment button to proceed to make a payment of $25 as a registration fee and finally to complete your account detail.

Once you are a registered user at Google Play, you can upload release-ready APK

Thank You