

Unit 4

Storing the Data Persistently

Introduction

Storing data persistently in Android refers to the practice of saving data in a way that it remains accessible even after the application is closed or the device is turned off. There are several methods available for achieving persistent data storage in Android, each catering to different use cases and requirements. Here are some of the common methods:

1. **Shared Preferences:**
This method is used to store small amounts of key-value pairs, typically configuration settings or user preferences. Shared Preferences are stored in XML files and can be accessed throughout the application. They are commonly used for storing simple settings like user preferences, theme selection, etc.
2. **Internal Storage:**
Internal storage is used for saving private data on the device's internal memory. This method is useful for storing sensitive or private data that is specific to your application. Data stored here is not directly accessible to other applications.
3. **External Storage:**
External storage refers to storage like SD cards or USB drives that can be used to store large files or data that is shareable between apps. However, since external storage can be shared, you should be careful about storing sensitive information here.
4. **SQLite Database:**
SQLite is a lightweight relational database management system that is included with Android. It allows you to create, read, update, and delete structured data using SQL queries. This method is suitable for more complex data structures where you need efficient querying capabilities.
5. **Content Providers:**
Content Providers allow your application to share data with other applications. They are often used to provide structured access to data stored in databases or other persistent storage mechanisms. Content Providers are a way to expose your app's data to other apps in a controlled manner.

The choice of storage method depends on factors such as the size and complexity of the data, security requirements, sharing capabilities, and performance considerations. In most cases, applications use a combination of these storage methods to cater to different aspects of data persistence. It is important to choose the right method for your specific use case to ensure efficient and secure data storage and retrieval.

Shared Preferences

Shared Preferences are used to save and retrieve the primitive data types (integer, float, boolean, string, long) data in the form of key-value pairs from a file within an app's file structure.

Generally, the Shared Preferences object will point to a file that contains key-value pairs and provides a simple read and write methods to save and retrieve the key-value pairs from a file.

The Shared Preferences file is managed by an android framework and it can be accessed anywhere within the app to read or write data into the file, but it's not possible to access the file from any other app so it's secured.

The Shared Preferences are useful to store the small collection of key-values such as user's login information, app preferences related to users, etc. to maintain the state of the app, next time when they login again to the app.

Handle Shared Preferences

We can save the preferences data either in single or multiple files based on our requirements.

If we use a single file to save the preferences, then we need to use `getPreferences()` method to get the values from Shared Preferences file and for multiple files, we need to call a `getSharedPreferences()` method and pass a file name as a parameter.

Method	Description
<code>getPreferences()</code>	This method is for activity level preferences and each activity will have its own preference file and by default, this method retrieves a default shared preference file that belongs to the activity.
<code>getSharedPreferences()</code>	This method is useful to get the values from multiple shared preference files by passing the name as a parameter to identify the file. We can call this from any Context in our app.

If we are using single shared preference file for our activity, then we need to initialize the `SharedPreferences` object by using `getPreferences()` method like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
```

If we are using multiple shared preference files, then we need to initialize the `SharedPreferences` object by using the `getSharedPreferences()` method like as shown below.

```
SharedPreferences pref = getSharedPreferences("file_name",MODE_PRIVATE);
```

Here, the name "file_name" is the preference file, which wants to read the values based on our requirements and the context mode `MODE_PRIVATE`, will make sure that the file can be accessed only within our application.

Write to Shared Preferences

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putBoolean("keyname",true);
editor.putString("keyname","string value");
editor.putInt("keyname","int value");
editor.putFloat("keyname","float value");
editor.putLong("keyname","long value");
editor.commit();
```

Read from Shared Preferences

To read or retrieve values from the Shared Preferences file, we need to call methods such as `getInt()`, `getString()`, etc. by providing the key for the value which we want to get like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
pref.getString("keyname", null);
pref.getInt("keyname", 0);
pref.getFloat("keyname", 0);
pref.getBoolean("keyname", true);
pref.getLong("keyname", 0);
```

Deleting from Shared Preferences

To delete values from the Shared Preferences file, we need to call `remove()` method by providing the key for the value which we want to delete like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.remove("keyname");
editor.commit();
```

Here, I have deleted the values from shared preferences using a method called `remove()` by providing the key for the value which we want to delete and committing the changes to shared preferences file using `commit()` method.

Clearing from Shared Preferences

We can clear all the data from Shared Preferences file using a `clear()` method like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.clear();
editor.commit();
```

Here, I have clear all the values from shared preferences using a method called `clear()` and committing the changes to shared preferences file using `commit()` method.

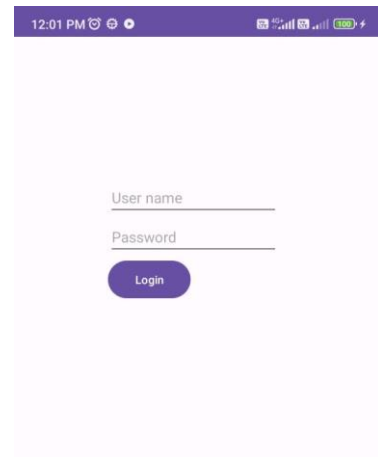
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText
        android:id="@+id/txtName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="150dp">
```

```

        android:layout_marginLeft="100dp"
        android:hint="User name"
        android:ems="10"/>
    <EditText
        android:id="@+id/txtPwd"
        android:inputType="textPassword"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Password"
        android:layout_marginLeft="100dp"
        android:ems="10" />
    <Button
        android:id="@+id/btnLogin"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:text="Login" />
</LinearLayout>

```



MainActivity.java

```

package com.example.sharedpreferencesexample;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    EditText uname, pwd;
    Button loginBtn;
    SharedPreferences pref;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        uname = (EditText)findViewById(R.id.txtName);
        pwd = (EditText)findViewById(R.id.txtPwd);
        loginBtn = (Button)findViewById(R.id.btnLogin);
        pref = getSharedPreferences("user_details",MODE_PRIVATE);
        intent = new Intent(MainActivity.this,DetailsActivity.class);
        if(pref.contains("username") && pref.contains("password")){

```

```

        startActivity(intent);
    }
    loginBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String username = uname.getText().toString();
            String password = pwd.getText().toString();
            if(username.equals("Harsh") && password.equals("hk")){
                SharedPreferences.Editor editor = pref.edit();
                editor.putString("username",username);
                editor.putString("password",password);
                editor.commit();
                Toast.makeText(getApplicationContext(), "Login
Successful",Toast.LENGTH_SHORT).show();
                startActivity(intent);
            }
            else
            {
                Toast.makeText(getApplicationContext(),"Credentials are
not valid",Toast.LENGTH_SHORT).show();
            }
        }
    });
}
}

```

activity_details.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/resultView"
        android:layout_gravity="center"
        android:layout_marginTop="170dp"
        android:textSize="20dp"/>
    <Button
        android:id="@+id/btnLogOut"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:text="Log Out" />
</LinearLayout>

```

DetailsActivity.java

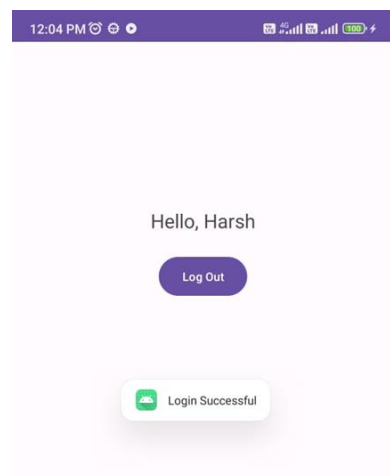
```
package com.example.sharedpreferencesexample;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class DetailsActivity extends AppCompatActivity {
    SharedPreferences prf;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details);
        TextView result = (TextView)findViewById(R.id.resultView);
        Button btnLogOut = (Button)findViewById(R.id.btnLogOut);
        prf = getSharedPreferences("user_details",MODE_PRIVATE);
        intent = new Intent(DetailsActivity.this,MainActivity.class);
        result.setText("Hello, "+prf.getString("username",null));
        btnLogOut.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SharedPreferences.Editor editor = prf.edit();
                editor.clear();
                editor.commit();
                startActivity(intent);
            }
        });
    }
}
```

Project: **SHAREDREFERENCESEXAMPLE**

Here, I have entered **Harsh** as username and **hk** as password that matches then we are redirecting user to another activity file to show the user details from shared preferences file. After that, if I am clicking on the Logout button, it will clear all the values in the shared preferences file and it will redirect user to login page.



Android Internal Storage

Internal Storage is useful to store the data files locally on the device's internal memory using a `FileOutputStream` object. After storing the data files in device internal storage, we can read the data file from the device using a `FileInputStream` object.

The data files saved in the internal are managed by an android framework and it can be accessed anywhere within the app to read or write data into the file, but it's not possible to access the file from any other app so it's secured. When the user uninstalls the app, automatically these data files will be removed from the device internal storage.

Write a File to Internal Storage

By using the android `FileOutputStream` object `openFileOutput` method, we can easily create and write data to a file in a device's internal storage.

```
String FILENAME = "GCET";  
String name = "Harsh"  
  
FileOutputStream fstream = openFileOutput(FILENAME, Context.MODE_PRIVATE);  
fstream.write(name.getBytes());  
fstream.close();
```

Here, I have created and writ in a file in device internal storage by using `FileOutputStream` object `openFileOutput` method with file name and `MODE_PRIVATE` mode to make the file private to our application. We used `write()` method to write the data in file and used `close()` method to close the stream.

In android, we have different modes such as `MODE_APPEND`, `MODE_WORLD_READABLE`, `MODE_WORLD_WRITEABLE`, etc. to use it in our application based on your requirements.

Read a File from Internal Storage

By using the android `FileInputStream` object `openFileInput` method, we can easily read the file from the device's internal storage.

```
String FILENAME = "GCET";  
FileInputStream fstream = openFileInput(FILENAME);  
StringBuffer sbuffer = new StringBuffer();  
int i;  
while ((i = fstream.read()) != -1){  
    sbuffer.append((char)i);  
}  
fstream.close();
```

Different Methods

- `getChannel()`: It returns the unique `FileChannel` object associated with this file output stream.
- `getFD()`: It returns the file descriptor which is associated with the stream.
- `write(byte[] b, int off, int len)`: It writes `len` bytes from the specified byte array starting at offset `off` to the file output stream.

- `read(byte[] b, int off, int len)`: It reads `len` bytes of data from the specified file input stream into an array of bytes.

When to Use Internal Storage:

- Store user-specific settings, preferences, or configuration files.
- Cache temporary data that can be regenerated if needed.
- Store app-specific private data that should not be accessible to other apps or users.

When Not to Use Internal Storage:

- Large media files or data that could consume a significant amount of storage space.
- Data that needs to be shared with other apps or users.

For larger files or data that should be accessible by other apps, you should consider using external storage options or cloud-based solutions. Keep in mind that Android offers other storage options like external storage (SD card), databases, and content providers, each with its own use cases and considerations.

Different classes and Methods

1. `Context`: The `'Context'` class is a fundamental class in Android that provides access to application-specific resources and services. To access internal storage, you typically need a `'Context'` object, which can be obtained from various components like `'Activity'`, `'Service'`, or `'Application'`.
2. `File`: The `'File'` class represents a file or directory in the file system. It is commonly used for file-related operations like creating, deleting, checking existence, and getting file properties.
3. `FileOutputStream`: This class is used for writing data into a file. It is typically used when you want to save data to a file in the internal storage.
4. `FileInputStream`: This class is used for reading data from a file. It is typically used when you want to retrieve data from a file in the internal storage.
5. `Context.getFilesDir()`: This method returns the absolute path to the directory on the filesystem where files created with `'openFileOutput()'` are stored. This is the primary directory for an app's private data.
6. `Context.openFileOutput()`: This method is used to open a private file associated with the current context. It returns a `FileOutputStream` that allows you to write data to the file.
7. `Context.openFileInput()`: This method is used to open a private file associated with the current context for reading. It returns a `FileInputStream` that allows you to read data from the file.
8. `File.delete()`: This method is used to delete a file or directory from the internal storage.
9. `File.list()`: This method returns an array of strings naming the files and directories in the internal storage directory represented by the `'File'` object.
10. `File.exists()`: This method is used to check if a file or directory exists in the internal storage.

activity_main.xml

```
<?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:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="50dp"
        android:textStyle="bold"
        android:textSize="25dp"
        android:text="Internal Storage Example" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="2in"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:layout_gravity="center"
        android:hint="File Name"
        android:ems="10"/>

    <EditText
        android:id="@+id/editText2"
        android:layout_width="2in"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:layout_gravity="center"
        android:hint="Data"
        android:ems="10" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:layout_gravity="center"
        android:text="Save" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="24dp"
        android:text="Read" />

        <TextView
            android:id="@+id/viewdata"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="175dp"
            android:layout_marginTop="400dp" />
    </LinearLayout>
```

MainActivity.java

```
package com.example.internalstorage;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

public class MainActivity extends AppCompatActivity {
    EditText editTextFileName,editTextData;

    TextView tv;
    Button saveButton,readButton;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        editTextFileName=findViewById(R.id.editText1);
        editTextData=findViewById(R.id.editText2);
        saveButton=findViewById(R.id.button1);
        readButton=findViewById(R.id.button2);
        tv=findViewById(R.id.viewdata);

        saveButton.setOnClickListener(new View.OnClickListener(){
```

```

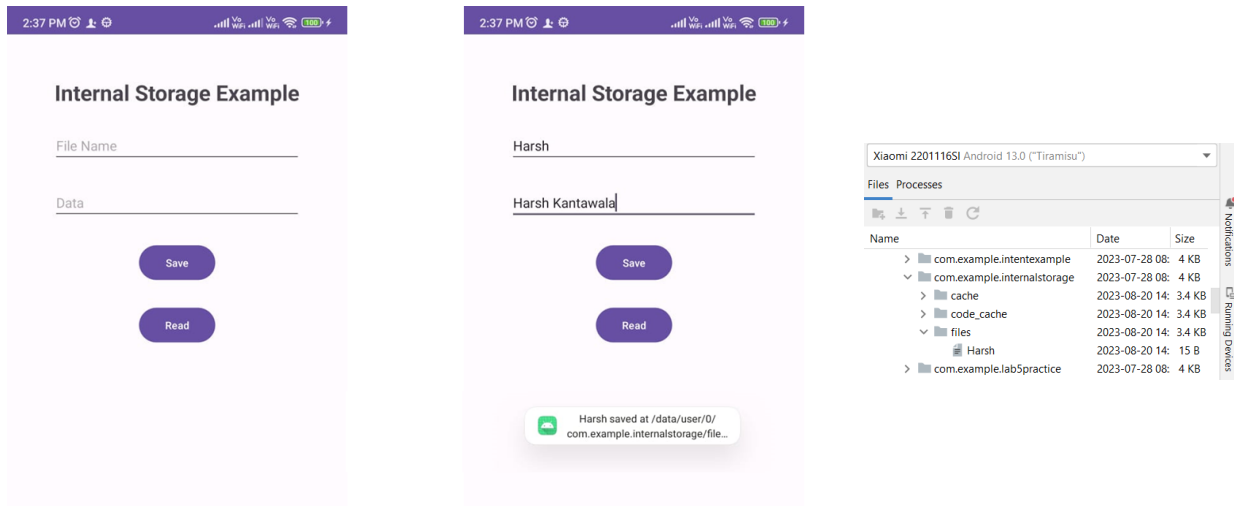
@Override
public void onClick(View arg0) {
    String filename=editTextFileName.getText().toString();
    String data=editTextData.getText().toString();

    FileOutputStream fos;
    try {
        fos = openFileOutput(filename, Context.MODE_PRIVATE);
        fos.write(data.getBytes());
        fos.close();
        Toast.makeText(getApplicationContext(),filename + "
saved" + " at " +getFilesDir() +"/",Toast.LENGTH_LONG).show();
    }
    catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}

});

readButton.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View arg0) {
        String filename=editTextFileName.getText().toString();
        StringBuffer stringBuffer = new StringBuffer();
        try {
            BufferedReader inputReader = new BufferedReader(new
InputStreamReader(openFileInput(filename)));
            String inputString;
            while ((inputString =
inputReader.readLine()) != null) {
                stringBuffer.append(inputString + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        Toast.makeText(getApplicationContext(),stringBuffer.toString(),Toast.LENGTH
_LONG).show();
    }
});
}
}

```



Project: INTERNALSTORAGE

Android External Storage

Here I will show how to use the External Storage option to store and retrieve the data from external storage media such as SD card.

External Storage is useful to store the data files publicly on the shared external storage using the `FileOutputStream` object. After storing the data files on external storage, we can read the data file from external storage media using a `FileInputStream` object.

The data files saved in external storage are word-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

Grant Access to External Storage

To read or write files on the external storage, our app must acquire the we have to give following permissions in `AndroidManifest.xml` file.

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

Checking External Storage Availability

Before we do any work with external storage, first we need to check whether the media is available or not by calling `getExternalStorageState()`. The media might be mounted to a computer, missing, read-only or in some other state. To get the media status, we need to write the code like as shown below.

```
boolean Available= false;
boolean Readable= false;
String state = Environment.getExternalStorageState();
if(Environment.MEDIA_MOUNTED.equals(state)){
    // Both Read and write operations available
    Available= true;
```

```
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)){  
    // Only Read operation available  
    Available= true;  
    Readable= true;  
} else {  
    // SD card not mounted  
    Available = false;  
}
```

By using `getExternalStoragePublishDirectory()` method we can access the files from appropriate public directory by passing the type of directory we want, such as `DIRECTORY_MUSIC`, `DIRECTORY_PICTURES`, `DIRECTORY_RINGTONES`, etc. based on our requirements.

If we save our files to corresponding media-type public directory, the system's media scanner can properly categorize our files in the system.

Write a File to External Storage

By using android `FileOutputStream` object and `getExternalStoragePublicDirectory` method, we can easily create and write data to the file in external storage public folders.

Writing a file in device public Downloads folder

```
String FILENAME = "GCET";  
String name = "Harsh";  
  
File folder =  
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);  
File myFile = new File(folder, FILENAME);  
FileOutputStream fstream = new FileOutputStream(myFile);  
fstream.write(name.getBytes());  
fstream.close();
```

Read a File from External Storage

By using the android `FileInputStream` object and `getExternalStoragePublicDirectory` method, we can easily read the file from external storage.

```
String FILENAME = "GCET";  
File folder = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);  
File myFile = new File(folder, FILENAME);  
FileInputStream fstream = new FileInputStream(myFile);  
StringBuffer sbuffer = new StringBuffer();  
int i;  
while ((i = fstream.read()) != -1){  
    sbuffer.append((char)i);  
}  
fstream.close();
```

Remember that external storage can be removable (e.g., SD card) or non-removable (e.g., internal shared storage on some devices). The approach you take for accessing external storage should be done carefully, considering various Android versions and storage configurations.

Also, keep in mind that best practices change over time, so be sure to refer to the official Android documentation for the most up-to-date methods and recommendations regarding external storage access.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="External Storage Example"
        android:textStyle="bold"
        android:textSize="25dp"
        android:layout_marginTop="50dp"
        android:layout_gravity="center"/>
    <EditText
        android:id="@+id/txtFileName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="150dp"
        android:layout_gravity="center"
        android:hint="File Name"
        android:ems="10" />
    <EditText
        android:id="@+id/txtName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:hint="Name"
        android:ems="10" />
    <EditText
        android:id="@+id/txtPhone"
        android:inputType="number"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Mobile"
        android:layout_gravity="center"
        android:ems="10" />
    <EditText
        android:id="@+id/txtEmail"
        android:inputType="textEmailAddress"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Email"
        android:layout_gravity="center">
```

```
        android:ems="10" />
    <Button
        android:id="@+id/btnSave"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Save" />
</LinearLayout>
```

MainActivity.java

```
package com.example.externalstorage;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity {

    EditText name, phone, email, fileNameEditText;
    Button saveBtn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        name = findViewById(R.id.txtName);
        phone = findViewById(R.id.txtPhone);
        email = findViewById(R.id.txtEmail);
        fileNameEditText = findViewById(R.id.txtFileName);

        saveBtn = findViewById(R.id.btnSave);

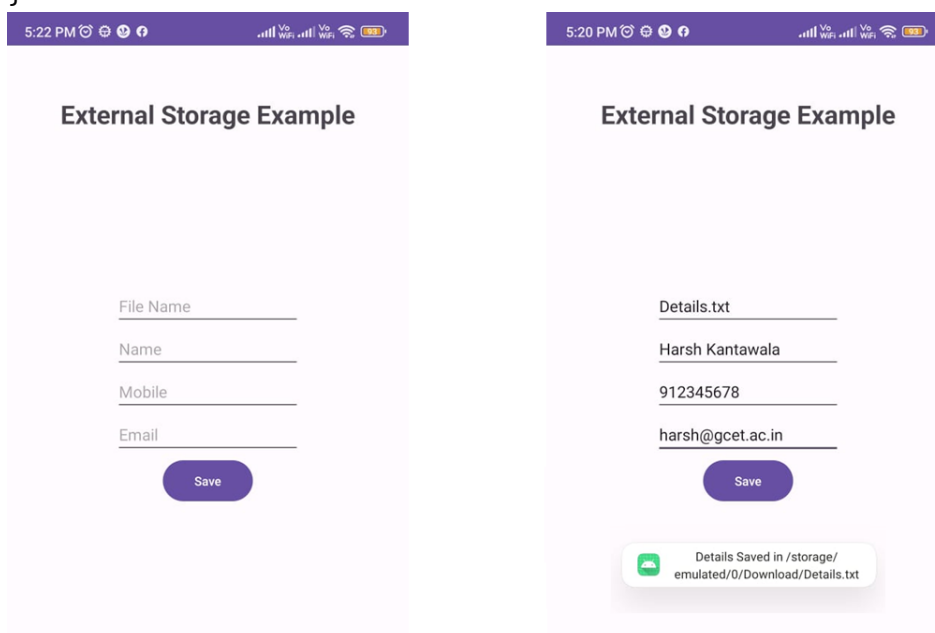
        saveBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String Name = name.getText().toString() + "\n";
                String Phone = phone.getText().toString() + "\n";
```

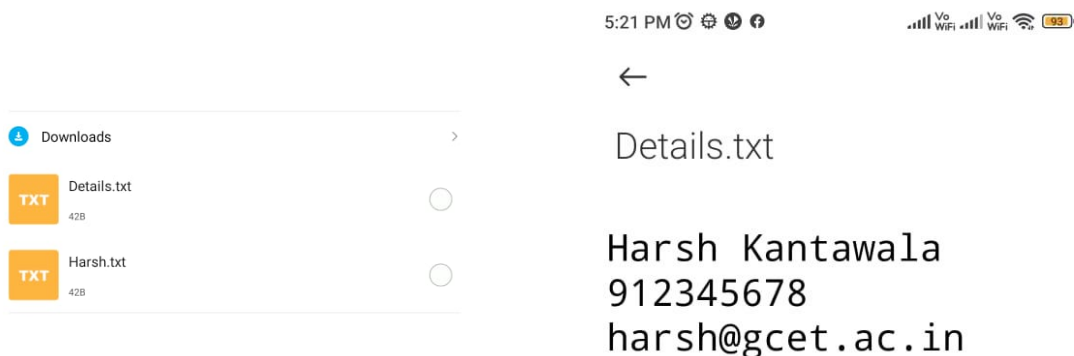
```
String Email = email.getText().toString();
try {
    ActivityCompat.requestPermissions(MainActivity.this,
        new
String[]{android.Manifest.permission.WRITE_EXTERNAL_STORAGE}, 1);

    String fileName =
fileNameEditText.getText().toString();
    File folder =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);

    File myFile = new File(folder, fileName);
    FileOutputStream fstream = new
FileOutputStream(myFile);
    fstream.write(Name.getBytes());
    fstream.write(Phone.getBytes());
    fstream.write(Email.getBytes());
    fstream.close();

    Toast.makeText(getApplicationContext(), "Details Saved
in " + myFile.getAbsolutePath(),
        Toast.LENGTH_SHORT).show();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
});
}
```





Project: **EXTERNALSTORAGE**

Android SQLite Database

Introduction about how to use the SQLite Database option to store structured data in a private database.

SQLite is an open-source lightweight relational database management system (RDBMS) to perform database operations, such as storing, updating, retrieving data from the database.

Generally, in our android applications Shared Preferences, Internal Storage and External Storage options are useful to store and maintain a small amount of data. In case, if we want to deal with large amounts of data, then SQLite database is the preferable option to store and maintain the data in a structured format.

By default, Android comes with built-in SQLite Database support so we do not need to do any configurations.

Just like we save the files on the device's internal storage, Android stores our database in a private disk space that is associated with our application and the data is secure, because by default this area is not accessible to other applications.

The package `android.database.sqlite` contains all the required APIs to use an SQLite database in our android applications.

Create Database and Tables using SQLite Helper

By using `SQLiteOpenHelper` class we can easily create the required database and tables for our application. To use `SQLiteOpenHelper`, we need to create a subclass that overrides the `onCreate()` and `onUpgrade()` call-back methods.

Methods:

`onCreate()`: This method is called only once throughout the application after the database is created and the table creation statements can be written in this method.

`onUpgrade()`: This method is called whenever there is an updation in the database like modifying the table structure, adding constraints to the database, etc.

We can insert data into the SQLite database by passing ContentValues to insert() method.

We can read the data from the SQLite database using the query() method in android applications.

We can update the data in the SQLite database using an update() method in android applications.

We can delete data from the SQLite database using the delete() method in android applications.

To implement SQLite database related activities we have to use java class file.

MyDBHandler.java

```
package com.example.practice;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import java.util.ArrayList;

public class MyDBHandler extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "studentsDB.db";
    private static final String TABLE_NAME = "students";
    private static final String COLUMN_ID = "id";
    private static final String COLUMN_NAME = "name";
    private static final String COLUMN_MOBILE = "mobile";
    private static final String COLUMN_EMAIL = "email";
    private static final String COLUMN_BRANCH = "branch";
    private static final String COLUMN_GENDER = "gender";

    public MyDBHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_TABLE_QUERY = "CREATE TABLE " + TABLE_NAME + "("
            + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
            + COLUMN_NAME + " TEXT,"
            + COLUMN_MOBILE + " TEXT,"
            + COLUMN_EMAIL + " TEXT,"
            + COLUMN_BRANCH + " TEXT,"
            + COLUMN_GENDER + " TEXT"
            + ")";
        db.execSQL(CREATE_TABLE_QUERY);
    }

    @Override
```

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
    onCreate(db);
}

public void addStudent(Student student) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COLUMN_NAME, student.getName());
    values.put(COLUMN_MOBILE, student.getMobile());
    values.put(COLUMN_EMAIL, student.getEmail());
    values.put(COLUMN_BRANCH, student.getBranch());
    values.put(COLUMN_GENDER, student.getGender());
    db.insert(TABLE_NAME, null, values);
    db.close();
}

public Student getStudentById(int studentId) {
    SQLiteDatabase db = this.getReadableDatabase();
    String[] columns = {
        COLUMN_ID,
        COLUMN_NAME,
        COLUMN_MOBILE,
        COLUMN_EMAIL,
        COLUMN_BRANCH,
        COLUMN_GENDER
    };

    String selection = COLUMN_ID + " = ?";
    String[] selectionArgs = {String.valueOf(studentId)};

    Cursor cursor = db.query(
        TABLE_NAME,
        columns,
        selection,
        selectionArgs,
        null,
        null,
        null
    );

    if (cursor != null && cursor.moveToFirst()) {
        int id =
        cursor.getInt(cursor.getColumnIndexOrThrow(COLUMN_ID));
        String name =
        cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_NAME));
    }
}
```

```
        String mobile =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_MOBILE));
        String email =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_EMAIL));
        String branch =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_BRANCH));
        String gender =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_GENDER));

        cursor.close();
        return new Student(id, name, mobile, email, branch, gender);
    }

    return null;
}

public ArrayList<Student> getAllStudents() {
    ArrayList<Student> studentList = new ArrayList<>();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery("SELECT * FROM " + TABLE_NAME, null);

    while (cursor.moveToNext()) {
        int id =
cursor.getInt(cursor.getColumnIndexOrThrow(COLUMN_ID));
        String name =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_NAME));
        String mobile =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_MOBILE));
        String email =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_EMAIL));
        String branch =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_BRANCH));
        String gender =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_GENDER));

        studentList.add(new Student(id, name, mobile, email, branch,
gender));
    }

    cursor.close();
    return studentList;
}

public int updateStudent(Student student) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COLUMN_NAME, student.getName());
}
```

```
values.put(COLUMN_MOBILE, student.getMobile());
values.put(COLUMN_EMAIL, student.getEmail());
values.put(COLUMN_BRANCH, student.getBranch());
values.put(COLUMN_GENDER, student.getGender());

return db.update(
    TABLE_NAME,
    values,
    COLUMN_ID + " = ?",
    new String[]{String.valueOf(student.getId())}
);
}

public void deleteStudent(int studentId) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(
        TABLE_NAME,
        COLUMN_ID + " = ?",
        new String[]{String.valueOf(studentId)}
    );
    db.close();
}
}
```

MainActivity.java

```
package com.example.practice;

import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {
    private EditText editTextId; // New field for entering student ID
    private EditText editTextName;
    private EditText editTextMobile;
    private EditText editTextEmail;
    private EditText editTextBranch;
    private RadioGroup radioGroupGender;
    private Button btnInsert;
    private Button btnUpdate;
}
```

```
private Button btnDelete;
private Button btnView;
private Button btnViewDetails;
private TextView textViewStudentData;
private MyDBHandler dbHandler;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    editTextId = findViewById(R.id.editTextId);
    editTextName = findViewById(R.id.editTextName);
    editTextMobile = findViewById(R.id.editTextMobile);
    editTextEmail = findViewById(R.id.editTextEmail);
    editTextBranch = findViewById(R.id.editTextBranch);
    radioGroupGender = findViewById(R.id.radioGroupGender);
    btnInsert = findViewById(R.id.btnInsert);
    btnUpdate = findViewById(R.id.btnUpdate);
    btnDelete = findViewById(R.id.btnDelete);
    btnView = findViewById(R.id.btnView);
    btnViewDetails = findViewById(R.id.btnViewDetails);
    textViewStudentData = findViewById(R.id.textViewStudentData);

    dbHandler = new MyDBHandler(this);

    btnInsert.setOnClickListener(view -> insertStudent());
    btnUpdate.setOnClickListener(view -> updateStudent());
    btnDelete.setOnClickListener(view -> deleteStudent());
    btnView.setOnClickListener(view -> viewStudents());
    btnViewDetails.setOnClickListener(view -> viewStudentDetails());
}

private void insertStudent() {
    String name = editTextName.getText().toString().trim();
    String mobile = editTextMobile.getText().toString().trim();
    String email = editTextEmail.getText().toString().trim();
    String branch = editTextBranch.getText().toString().trim();

    int selectedGenderId = radioGroupGender.getCheckedRadioButtonId();
    if (selectedGenderId == -1) {
        Toast.makeText(MainActivity.this, "Please select a gender.",
            Toast.LENGTH_SHORT).show();
        return;
    }
    RadioButton selectedGenderRadioButton =
        findViewById(selectedGenderId);
```

```
String gender = selectedGenderRadioButton.getText().toString();

    if (name.isEmpty() || mobile.isEmpty() || email.isEmpty() ||
branch.isEmpty()) {
        Toast.makeText(MainActivity.this, "Please fill all fields.",
Toast.LENGTH_SHORT).show();
        return; // Return early to prevent insertion
    }

    Student newStudent = new Student(name, mobile, email, branch,
gender);
    dbHelper.addStudent(newStudent);

    editTextName.setText("");
    editTextMobile.setText("");
    editTextEmail.setText("");
    editTextBranch.setText("");
    radioGroupGender.clearCheck();

    Toast.makeText(MainActivity.this, "Student added successfully.",
Toast.LENGTH_SHORT).show();
}

private void updateStudent() {
    String idText = editTextId.getText().toString().trim();

    if (idText.isEmpty()) {
        Toast.makeText(MainActivity.this, "Please enter the Student
ID.", Toast.LENGTH_SHORT).show();
        return;
    }

    int studentIdToUpdate = Integer.parseInt(idText);

    String name = editTextName.getText().toString().trim();
    String mobile = editTextMobile.getText().toString().trim();
    String email = editTextEmail.getText().toString().trim();
    String branch = editTextBranch.getText().toString().trim();

    int selectedGenderId = radioGroupGender.getCheckedRadioButtonId();
    RadioButton selectedGenderRadioButton =
findViewById(selectedGenderId);
    String gender = selectedGenderRadioButton.getText().toString();

    if (name.isEmpty() || mobile.isEmpty() || email.isEmpty() ||
branch.isEmpty()) {
        Toast.makeText(MainActivity.this, "Please fill all fields.",
```

```
Toast.LENGTH_SHORT).show();
    return;
}

Student updatedStudent = new Student(studentIdToUpdate, name,
mobile, email, branch, gender);
dbHandler.updateStudent(updatedStudent);

editTextId.setText("");
editTextName.setText("");
editTextMobile.setText("");
editTextEmail.setText("");
editTextBranch.setText("");
radioGroupGender.clearCheck();

Toast.makeText(MainActivity.this, "Student updated successfully.",
Toast.LENGTH_SHORT).show();
}

private void deleteStudent() {
    String idText = editTextId.getText().toString().trim();

    if (idText.isEmpty()) {
        Toast.makeText(MainActivity.this, "Please enter the Student
ID.", Toast.LENGTH_SHORT).show();
        return;
    }

    int studentIdToDelete = Integer.parseInt(idText);
    dbHandler.deleteStudent(studentIdToDelete);

    editTextId.setText("");
    editTextName.setText("");
    editTextMobile.setText("");
    editTextEmail.setText("");
    editTextBranch.setText("");
    radioGroupGender.clearCheck();

    Toast.makeText(MainActivity.this, "Student deleted successfully.",
Toast.LENGTH_SHORT).show();
}

private void viewStudents() {
    ArrayList<Student> studentList = dbHandler.getAllStudents();

    StringBuilder stringBuilder = new StringBuilder();
    for (Student student : studentList) {
```



```
        stringBuilder.append(student.toString()).append("\n\n");
    }

    if (studentList.isEmpty()) {
        textViewStudentData.setText("No student records found.");
    } else {
        textViewStudentData.setText(stringBuilder.toString());
    }
}

private void viewStudentDetails() {
    String idText = editTextId.getText().toString().trim();

    if (idText.isEmpty()) {
        textViewStudentData.setText("Please enter the Student ID.");
        return;
    }

    int studentIdToView = Integer.parseInt(idText);
    Student student = dbHelper.getStudentById(studentIdToView);

    if (student == null) {
        textViewStudentData.setText("No student record found with the
provided ID.");
    } else {
        String studentDetails = student.toString();
        textViewStudentData.setText(studentDetails);
    }
}
}
```

Student.java

```
package com.example.practice;

public class Student {
    private int id;
    private String name;
    private String mobile;
    private String email;
    private String branch;
    private String gender;

    public Student() {
    }

    public Student(String name, String mobile, String email, String branch,
String gender) {
```

```
        this.name = name;
        this.mobile = mobile;
        this.email = email;
        this.branch = branch;
        this.gender = gender;
    }
    public Student(int id, String name, String mobile, String email, String
branch, String gender) {
        this.id = id;
        this.name = name;
        this.mobile = mobile;
        this.email = email;
        this.branch = branch;
        this.gender = gender;
    }
    // Getters and Setters for all fields
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getMobile() {
        return mobile;
    }
    public void setMobile(String mobile) {
        this.mobile = mobile;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getBranch() {
        return branch;
    }
    public void setBranch(String branch) {
        this.branch = branch;
    }
    public String getGender() {
```

```

        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
    @Override
    public String toString() {
        return "ID: " + id
            + "\nName: " + name
            + "\nMobile: " + mobile
            + "\nEmail: " + email
            + "\nBranch: " + branch
            + "\nGender: " + gender;
    }
}

```

activity_main.xml

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number"
        android:hint="Enter Student ID (for Update/Delete)"/>

    <EditText
        android:id="@+id/editTextName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Name"/>

    <EditText
        android:id="@+id/editTextMobile"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number"
        android:hint="Mobile"/>

    <EditText
        android:id="@+id/editTextEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress"

```

```
        android:hint="Email"/>

<EditText
    android:id="@+id/editTextBranch"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Branch"/>

<RadioGroup
    android:id="@+id/radioGroupGender"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_gravity="center">

    <RadioButton
        android:id="@+id/radioButtonMale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Male"/>

    <RadioButton
        android:id="@+id/radioButtonFemale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Female"/>
</RadioGroup>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:orientation="horizontal">

    <Button
        android:id="@+id/btnInsert"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:text="Insert"/>

    <Button
        android:id="@+id/btnUpdate"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:text="Update"/>

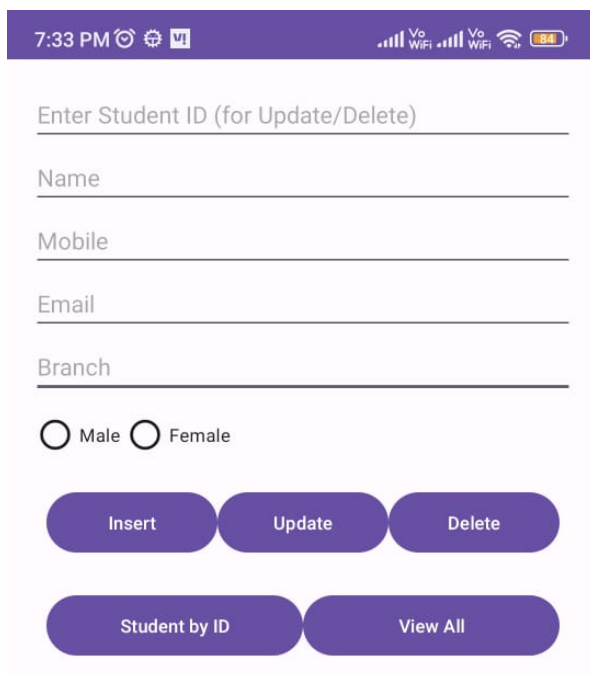
    <Button
        android:id="@+id/btnDelete"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:text="Delete"/>
```

```

</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:orientation="horizontal">
    <Button
        android:id="@+id/btnViewDetails"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:text="Student by ID"/>
    <Button
        android:id="@+id/btnView"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:text="View All"/>
</LinearLayout>

<TextView
    android:id="@+id/textViewStudentData"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Student Details:"
    android:textStyle="bold"
    android:textSize="16sp"
    android:paddingTop="16dp"/>
</LinearLayout>

```



7:33 PM

Enter Student ID (for Update/Delete)

Name

Mobile

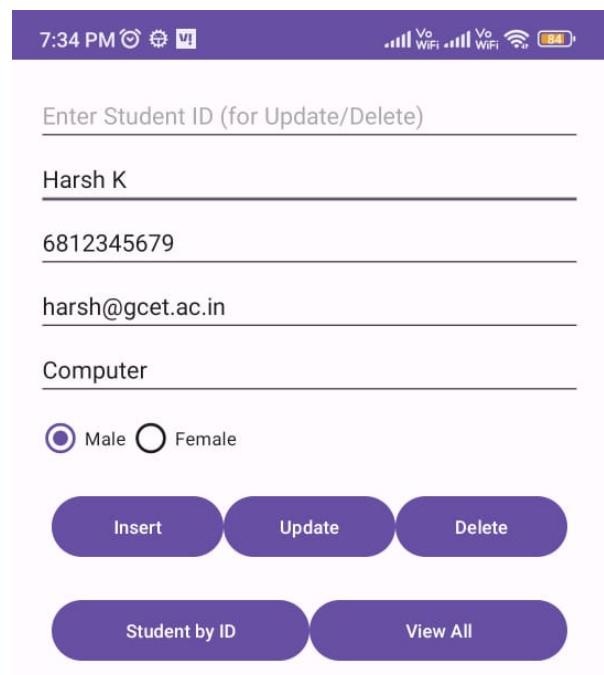
Email

Branch

☐ Male ☐ Female

Insert Update Delete

Student by ID View All



7:34 PM

Enter Student ID (for Update/Delete)

Harsh K

6812345679

harsh@gcet.ac.in

Computer

☒ Male ☐ Female

Insert Update Delete

Student by ID View All

7:34 PM

3

Name

Mobile

Email

Branch

☐ Male ☐ Female

Insert Update Delete

Student by ID View All

ID: 3
Name: Harsh
Mobile: 981234567
Email: harsh@gcet.ac.in
Branch: Anand
Gender: Male

7:35 PM

Enter Student ID (for Update/Delete)

Name

Mobile

Email

Branch

☐ Male ☐ Female

Insert Update Delete

Student by ID View All

ID: 1
Name: Abc
Mobile: 712348689
Email: abc@def.com
Branch: Anand
Gender: Male

ID: 2
Name: Xyz Pqr
Mobile: 812345769
Email: pqr@xyz.co.in
Branch: Bakrol
Gender: Female

Project: Practice

	id	name	mobile	email	branch	gender
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Abc	712348689	abc@def.com	Anand	Male
2	2	Xyz Pqr	812345769	pqr@xyz.co.in	Bakrol	Female
3	3	Harsh	981234567	harsh@gcet.ac.in	Anand	Male
4	4	Harsh K	6812345679	harsh@gcet.ac.in	Computer	Male

To download database go to `data>data>com.example.practice(projectname)>databases`. Save it and to view download DB Browser for sqlite.

Android Content Providers

Content Provider will act as a central repository to store the data of the application in one place and make that data available for different applications to access whenever it's required.

We can configure Content Providers to allow other applications securely access and modify our app data based on our requirements.

The Content Provider is a part of an android application and it will act like more like a relational database to store the app data. We can perform multiple operations like insert, update, delete and edit on the data stored in content provider using `insert()`, `update()`, `delete()` and `query()` methods.

We can use content provider whenever we want to share our app data with other apps and it allow us to make a modifications to our application data without effecting other applications which depends on our app.

Content provider is having different ways to store app data. The app data can be stored in a SQLite database or in files or even over a network based on our requirements. By using content providers we can manage data such as audio, video, images and personal contact information.

We have different type of access permissions available in content provider to share the data. We can set a restrict access permissions in content provider to restrict data access limited to only our application and we can configure different permissions to read or write a data.

Access Data from Content Provider

To access a data from content provider, we need to use ContentResolver object in our application to communicate with the provider as a client. The ContentResolver object will communicate with the provider object (ContentProvider) which is implemented by an instance of class.

Generally, in android to send a request from UI to ContentResolver we have another object called CursorLoader which is used to run the query asynchronously in background. In android application, the UI components such as Activity or Fragment will call a CursorLoader to query and get a required data from ContentProvider using ContentResolver.

The ContentProvider object will receive data requests from the client, performs the requested actions (create, update, delete, retrieve) and return the result.

Content URIs

Content URI is a URI that is used to query a content provider to get the required data. The Content URIs will contain the name of an entire-provider (authority) and the name that points to a table (path).

content://authority/path

Following are the details about various parts of an URI in android application.

content:// - The string content:// is always present in the URI and it is used to represent the given URI is a content URI.

authority - It represents the name of content provider, for example phone, contacts, etc. and we need to use a fully qualified name for third party content providers like com.example.cprovider

path - It represents the table's path.

The ContentResolver object use the URI's authority to find the appropriate provider and send the query objects to the correct provider. After that ContentProvider uses the path of content URI to choose the right table to access.

Following is the example of a simple example of URI in android applications.

content://contacts_info/users



Here the string content:// is used to represent URI is a content URI, contacts_info string is the name of the provider's authority and users string is the table's path.

Creating a Content Provider

To create a content provider in android applications we should follow below steps.

- We need to create a content provider class that extends the ContentProvider base class.
- We need to define our content provider URI to access the content.
- The ContentProvider class defines a six abstract methods (insert(), update(), delete(), query(), getType()) which we need to implement all these methods as a part of our subclass.
- We need to register our content provider in AndroidManifest.xml using <provider> tag.

Following are the list of methods which need to implement as a part of ContentProvider class.

Android Content Provider - Insert, Query, Update, Delete, getType, onCreate Methods

query() - It receives a request from the client. By using arguments it will get a data from the requested table and return the data as a Cursor object.

insert() - This method will insert a new row into our content provider and it will return the content URI for newly inserted row.

update() - This method will update existing rows in our content provider and it returns the number of rows updated.

delete() - This method will delete the rows in our content provider and it returns the number of rows deleted.

getType() - This method will return the MIME type of data to given content URI.

onCreate() - This method will initialize our provider. The android system will call this method immediately after it creates our provider.

Example

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:padding="16dp"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/contactListView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</RelativeLayout>
```


MainActivity.java

```
package com.example.cprovider;

import android.Manifest;
import android.annotation.SuppressLint;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.widget.AdapterView;
import android.widget.ListView;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class MainActivity extends AppCompatActivity {

    private static final int REQUEST_READ_CONTACTS = 101;
    private ListView contactListView;
    private ArrayList<String> contactList;
    private ArrayAdapter<String> adapter;

    /*Declare class-level variables for the activity, including a request
    code for
    the permission request, a ListView to display contacts, an ArrayList to
    store the contacts,
    and an ArrayAdapter to bind the data to the ListView.
    */

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        contactListView = findViewById(R.id.contactListView);
        contactList = new ArrayList<>();
        adapter = new ArrayAdapter<>(this,
        android.R.layout.simple_list_item_1, contactList);
        contactListView.setAdapter(adapter);

        // Check for permission and request if necessary
        if (ContextCompat.checkSelfPermission(this,
```

```

Manifest.permission.READ_CONTACTS)
    != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.READ_CONTACTS},
REQUEST_READ_CONTACTS);
    } else {
        loadContacts();
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, String[]
permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
    if (requestCode == REQUEST_READ_CONTACTS) {
        if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            loadContacts();
        } else {
        }
    }
}

/*The onRequestPermissionsResult() method is called when the user
responds to the
    permission request dialog. If the user grants the READ_CONTACTS
permission, call the loadContacts() method.
    If the user denies the permission, you can handle it appropriately
(e.g., show an error message).
*/
private void loadContacts() {
    contactList.clear();

    // Query contacts using the content provider
    Uri contactsUri =
ContactsContract.CommonDataKinds.Phone.CONTENT_URI;
    Cursor cursor = getContentResolver().query(contactsUri, null, null,
null, null);

    if (cursor != null) {
        while (cursor.moveToNext()) {
            @SuppressWarnings("Range") String contactName =
cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Pho
ne.DISPLAY_NAME));
            @SuppressWarnings("Range") String contactNumber =
cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Pho
ne.NUMBER));

```

```

        contactList.add(contactName + " - " + contactNumber);
    }
    cursor.close();
}

// Sort the contact names alphabetically
Collections.sort(contactList, String.CASE_INSENSITIVE_ORDER);

adapter.notifyDataSetChanged();
}

/*
1. The loadContacts() method is responsible for loading the contacts
from the device's contact provider and displaying them in the ListView.
2. It starts by clearing the contactList to ensure that we don't
duplicate contacts.
3. It then queries the contacts using the content provider.
    The ContactsContract.CommonDataKinds.Phone.CONTENT_URI represents
the URI for the contacts stored in the phone's contact database.
4. The Cursor is used to traverse the result of the query and extract
contact names and numbers.
5. The contact names and numbers are added to the contactList as
strings with the format "Name - Number."
6. After fetching the contacts, the contactList is sorted
alphabetically using Collections.sort() with String.CASE_INSENSITIVE_ORDER
as the comparator.
    This ensures that the names are sorted in a case-insensitive manner.
7. Finally, the adapter is notified of the data change using
adapter.notifyDataSetChanged(), which updates the ListView with the sorted
contact names.*/
}

```

Project: **C PROVIDER**

In this example, a content provider is used to access the device's contact data.

It demonstrates how to request and handle permissions, query the contact database, and use a `ListView` with an `ArrayAdapter` to display the fetched data.