Native Data Handling

Course: Mobile Application Development

Unit : III – Building Blocks of Mobile Apps - II

Class / Semester: II MCA / III Semester

Department of MCA

INTRODUCTION

- Scenarios where the app data may have to be stored permanently in order to be retrieve at later
- Data can be saved either locally on the device or remotely on the servers
- Data could be either primitive or complex in nature, and can be stored on the device in an unstructured or structured manner
- Android framework offers several options for persistence:
  - **SharedPreferences:** store primitive private data on key-value pairs
  - **Internal Storage:** store private data in the device memory
  - **External Storage:** store public data on the shared external storage
  - **SQLite Databases:** store structured data in a private database
  - **Network server** : store data on the remote web server

SHARED PREFERENCES

- This class allows you to save and retrieve key / value pairs of primitive data type such as ringtone, app setting etc..
- We use same for saving the primitive data: booleans, floats, ints, longs, and strings
- Data will persist in the user session
- Shared preferences stores data in an XML file in the internal memory of the device
- The creation, storage, and manipulation of the XML file are internally taken care by the SharedPreferences API
- To create this object, we use *getSharedPreferences (String name, int mode)*

SHARED PREFERENCES

- To write values,
  - Call the method edit () to get a SharedPreferences.Editor
  - Add values methods such as putBoolean(), putInt(), putFloat() and putString()
  - Persists the new values with commit()
- To read values,
  - use the methods as getBoolean () and getString ()

```
SharedPreferences preferences =
getSharedPreferences("SMSPreferences",MODE_PRIVATE);
 btnSave.setOnClickListener(new OnClickListener() {
 @Override
 public void onClick(View arg0) {
 Editor editor=preferences.edit();
 editor.putBoolean("SendSMS", chkEnable.isChecked());
 editor.putString("Message", etMessage.getText().toString());
 editor.putString("Signature",
etSignature.getText().toString());
editor.commit();
 }
 });
```

SHARED PREFERENCES

- To write values,
  - Call the method edit () to get a SharedPreferences.Editor
  - Add values methods such as putBoolean(), putInt(), putFloat() and putString()
  - Persists the new values with commit()

```
SharedPreferences preferences =
getSharedPreferences(“SMSPreferences”,MODE_PRIVATE);
 btnSave.setOnClickListener(new OnClickListener() {
 @Override
 public void onClick(View arg0) {
 Editor editor=preferences.edit();
 editor.putBoolean(“SendSMS”, chkEnable.isChecked());
 editor.putString(“Message”, etMessage.getText().toString());
 editor.putString(“Signature”,
etSignature.getText().toString());
editor.commit();
 }
 });
```

SHARED PREFERENCES

- To read values,
  - use the methods as getBoolean () and getString ()

```
private void sendSMS() {
 SharedPreferences preferences=
context.getSharedPreferences(“SMSPreferences”,
context.MODE_PRIVATE);
 boolean sendSms=preferences.getBoolean(“SendSMS”,
false);
 String message=preferences.getString(“Message”, “”);
 String signature=preferences.getString(“Signature”, “”);
 if(sendSms==true)
 {
//Send the SMS to the caller
 }
 }
```

INTERNAL STORAGE

- Files saved to the internal storage are deprived of their application, allowing other applications can not access them
- When the user uninstalls the app, these files are removed
- To create and save a private file to the internal storage
  - Call openFileOutput () with the file name and the operating mode (in case MODE_PRIVATE) which returns a FileOutputStream;
  - Write on file with the write ()
  - Close the stream with close ()

```
String FILENAME = "myfile";
String string = "hello world !";
FileOutputStream fos = openFileOutput(FILENAME,
Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

EXTERNAL STORAGE

- ❑ It may be removable storage media (such as an SD card) or an internal memory (not removable)
- ❑ Files saved to the external storage are reading for all and can be modified by the user when they allow USB mass storage to transfer files from a computer
- ❑ It should always call **Environment.getExternalStorageState ()** to check that the media is available before doing any work with external storage

EXTERNAL STORAGE – CHECK MEDIA

```
boolean mExtStorageAvailable = false;
boolean mExtStorageWriteable = false;
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state))
{
  mExtStorageAvailable = mExternalStorageWriteable = true; }
else
if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state))
{
  mExtStorageAvailable = true;
  mExtStorageWriteable = false;
 }
else
{
mExtStorageAvailable = mExtStorageWriteable = false;
}
```

EXTERNAL STORAGE – ACCESS FILES

- Use getExternalFilesDir() to open a File representing the external storage directory
- Method requires a parameter that specifies the type of sub-directory you want, such as: Environment.DIRECTORY_MUSIC and Environment.DIRECTORY_RINGTONES (null to receive the root of your application directory)
- This method will create the appropriate directory, if necessary.

```
File dir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
File file = new File(dir, "test.obj");
 FileOutputStream fos = new FileOutputStream(file); ObjectOutputStream oos = new
ObjectOutputStream(fos); oos.writeObject(objeto);
```

REFERENCES

- Anubhav Pradhan, Anil V Deshpande, "Composing Mobile Apps using Android", Wiley Edition, 2014
- https://www.tutorialspoint.com/android/android_application_components.htm
- https://www.javatpoint.com/android-core-building-blocks

SQLite

USE CASE PROBLEM

Association of one gated community decided to introduce system for visitor’s recording to the Apartment houses

- Security at the front gate is the end user
- Mobile notification sent to Resident/Host for approval
- Resident may accept/reject the visitor
- Security falls for manual checking if no response is received
- Pre-authorized guest provision may be given (Expected visitor)

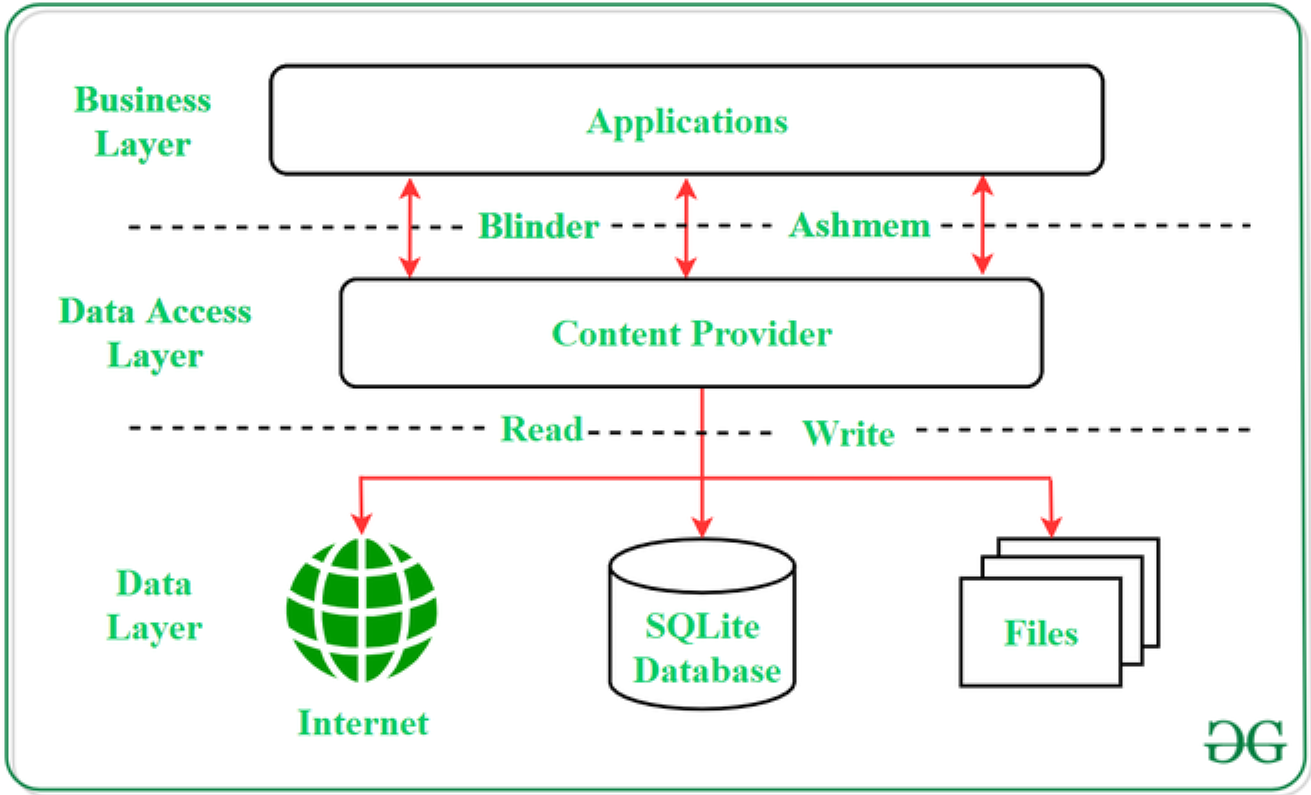
ANDROID – SQLITE DATABASE

- Open source RDBMS SQL database that stores data to a text file on a device
- Supports all the relational database features and available in android.database.sqlite
- Written in C, supports cross-mobile platform , configure it with less than 250 Kbs
- SQLite transactions are fully ACID(Atomicity, Consistency, Isolation, Durability)compliant
- Databases are stored in the /data/data/<package-name>/databases directory.
- Advantages
  - light weight database
  - Requires very little memory
  - Automatically managed database

android.database.sqlite Package

ANDROID – SQLITE DATABASE

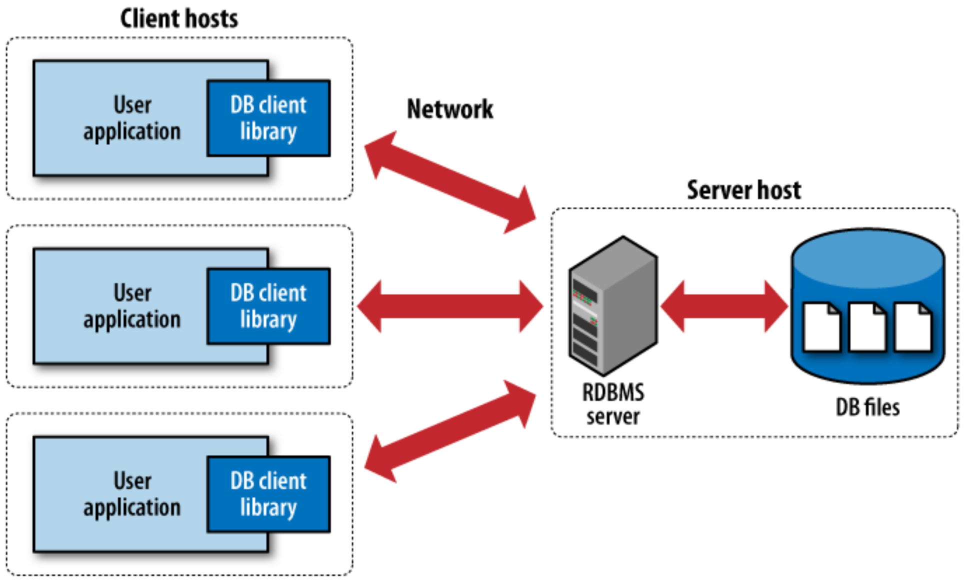
Relationship between layers to access Data



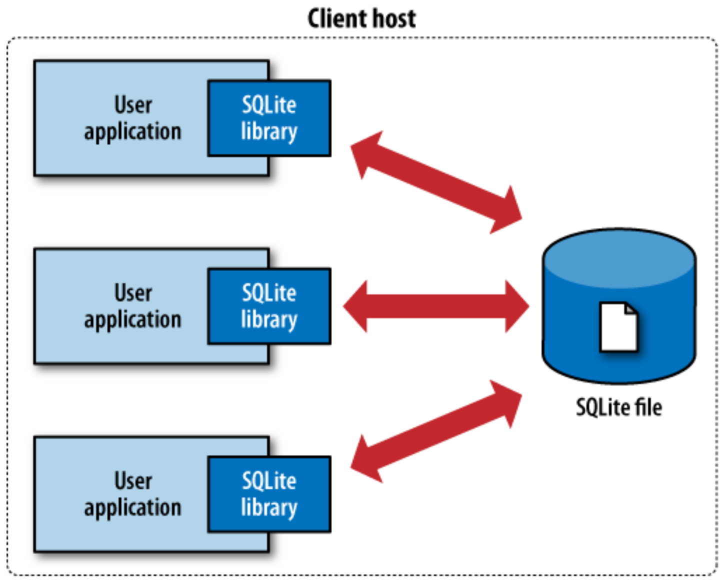
ANDROID – SQLITE DATABASE

- SQLite supports only 3 Datatypes
    - Text(like string)
    - Integer(like int)
    - Real(like double)
- android.database.sqlite.SQLiteOpenHelper class is used to manage database creation

How's SQLite different from traditional databases?



(a) Traditional client-server architecture



(b) SQLite serverless architecture

ANDROID – SQLITE DATABASE

- android.database.sqlite.SQLiteOpenHelper class is used to create and manage database

constructor
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)

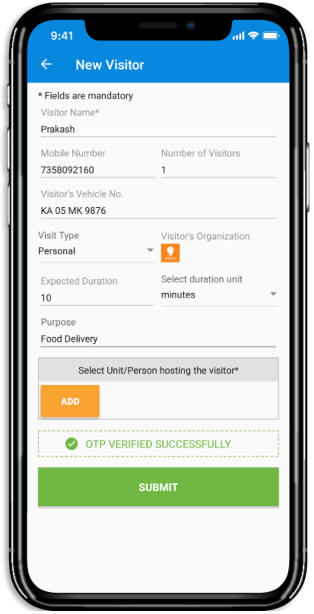
Methods
public abstract void onCreate(SQLiteDatabase db)
public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
public synchronized void close ()
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)

ANDROID – SQLITE DATABASE

- SQLiteDatabase class is used to perform actions on database

Methods
void execSQL(String sql)
long insert(String table, String nullColumnHack, ContentValues values)
int update(String table, ContentValues values, String whereClause, String[] whereArgs)
Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy
Int delete(String table, String whereClause, String[] whereArgs)
static boolean deleteDatabase(File file)
openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorhandler)

ANDROID – SQLITE DATABASE



9:41

New Visitor

* Fields are mandatory

Visitor Name*
Prakash

Mobile Number
7358092160

Number of Visitors
1

Visitor's Vehicle No.
KA 05 MK 9876

Visit Type
Personal

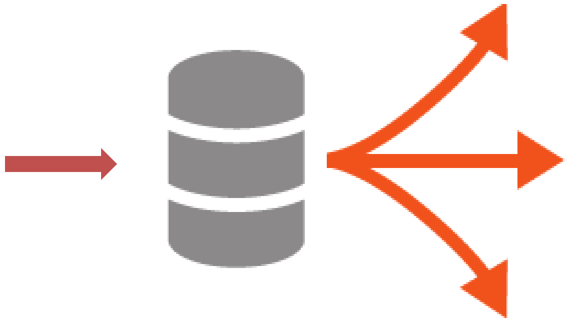
Visitor's Organization

Expected Duration
10

Select duration unit
minutes

Purpose
Food Delivery

Select Unit/Person hosting the visitor*



Date	Visitor Name	Mobile No	Apartment No.
01.01.2020	Priya	1231245	A24
01.01.2020	Riya	1231245	A12
01.01.2020	Sandy	1231245	C29

ANDROID – SQLITE DATABASE

- An alternative way of opening/creating a SQLITE database in your local Android's data space is given below

  SQLiteDatabase db = this.openOrCreateDatabase("myfriendsDB", MODE_PRIVATE, null);

- MODE could be: MODE_PRIVATE, MODE_WORLD_READABLE, and MODE_WORLD_WRITEABLE

SQLiteDatabase db = this.openOrCreateDatabase("myfriendsDB", MODE_PRIVATE, null);

MODE could be: MODE_PRIVATE, MODE_WORLD_READABLE, and MODE_WORLD_WRITEABLE. Meaningful for apps consisting of multiples activities

http://yuliana.lecturer.pens.ac.id/Android/Download/ppt/