Chapter 5

Android Data Storage

Android Data Storage

- **Android provides several options to save application data:**
  - **Shared Preference:**
    - store private primitive data in key-value pairs
  - **Internal Storage:**
    - store private data on device memory
  - **External Storage:**
    - store public data on shared external storage
  - **SQLite Database:**
    - store a structured data in a private database
  - **Network Connection:**
    - store data on the web with your network server

Shared Preference

- used to save and retrieve the primitive data types (integer, float, Boolean, string, long) data in the form of key-value pair from a file within an apps file structure. For example:-username-abc, password – abc123
- Shared Preferences can also be used to **manage session**. Session is a way of making some data available throughout an application.
- **Shared Preferences** object will point to a file that contains a key-value pairs and provides a simple read and write methods to save and retrieve the key-value pairs from a file.
- **Shared Preferences file**
  - is managed by an android framework
  - 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
  - Data is stored in XML file under **data/data/package-name/shared-prefs** folder

Accessing Shared Preference

- **SharedPreference** object provides two methods
  - **getSharedPreferences**( ):-useful to get the values from multiple shared preference files by passing the name as parameter to identify the file. It can be called from any Context in our app. For example

```
SharedPreferences sp=getSharedPreferences("filename1",Context.MODE_PRIVATE);
```

  - **getPreferences**():-for **activity level preferences** and each activity will have it's own preference file and by default this method retrieves a default shared preference file that belongs to the activity. For example

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

Write to Shared Preference

- We need an **editor** to edit and save the changes in **SharedPreferences** object. Example

```
SharedPreferences sp = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sp.edit();
editor.putString(“keyname",stringValue);
editor.putInt(“keyname",integerValue);
editor.commit();
```

Read from Shared Preference

```
SharedPreferences sp = getPreferences(Context.MODE_PRIVATE);
sp.getString("keyname",null);
sp.getInt("keyname",0);
```

Delete from Shared Preference

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

Clearing from Shared Preference

- We can clear all the data from Shared Preferences file

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

Internal Storage

- useful to store or retrieve data to/from the device's internal memory using **FileOutputStream/FileInputStream** object
- These files are **private** to your application and the files are removed when you **uninstall** the application.
- **To write a file to internal storage**
  - Call **openFileOutput(String filename, int mode).** The modes can be Context.MODE_PRIVATE, Context.MODE_APPEND
  - Write to the file with **write()** method
  - Close the stream with **close()** method

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

- **To read a file to internal storage**
  - Call **openFileInput(String filename)**.
  - Read bytes from the file with **read()** method. (since we read data byte by byte we have to append each character using **append()** method)
  - Close the stream with **close()** method

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

External Storage

- **To write data to external storage**
  - Add permissions(such as READ_EXTERNAL_STORAGE or WRITE_EXTERNAL_STORAGE) to read or write files on the external storage.

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

  - Check External Storage Availability:

```
public Boolean isExternalStorageWritable(){
 String state=Environment.getExternalStorageState();
if(Environment.MEDIA_MOUNTED.equals(state) )  return true;  }
return false;   }
public Boolean isExternalStorageReadable(){
 String state=Environment.getExternalStorageState();
if(Environment.MEDIA_MOUNTED.equals(state) || Environment.MEDIA_MOUNTED_RE
AD_ONLY.equals(state) )  return true;  }
return false;
 }
```

- Get the file directory by calling
  - **getExternalStorageFilesDir()-** create files specific to your application and removed when your application is uninstalled

File folder=getExternalStorageFilesDir("MyFolder");

Files myfile=new File(folder,"mydata.txt");

  - **getExternalStoragePublicDirectory()-** not specific to your application and not removed when your application is uninstalled. It includes directories like DIRECTORY_MUSIC, DIRECTORY_PICTURES,DIRECTORY_RINGTONES, DIRECTORY_DOWNLOADS

File folder=Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);

Files myfile=new File(folder,"mydata.txt");

- Use FileOutputStream object to write data:

FileOutputStream fstream = new FileOutputStream(myfile);

fstream.write(name.getBytes());
fstream.close();

- **To read data from external storage**
  - Get the file directory by calling
    - **getExternalStorageFilesDir() or**
    - **getExternalStoragePublicDirectory()**
  - Call openFileInputStream method by passing the file name as argument
  - Read bytes from the file using read() method
  - Close the stream using close() method

```
String FILENAME = “mydata.txt";
File folder = Environment.getExternalStoragePublicDirectory(Environment.DIR
ECTORY_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();
```

SQLite

- open source light weight RDBMS used to perform database operations
- **Advantage**
  - Serverless which means it is simple to set up and zero configuration is required
  - File based system makes it very portable (android stores our database in a private disk space that's associated to our application and the data is secure)
  - Great for development and testing
- **Disadvantage**
  - Does not provide network access (i.e. accessing it from another machine)
  - Not built for large-scale applications
  - No user management
- **android.database.sqlite** contains all the required API's to use SQLite database

Creating Database and Table

- by using **SQLiteOpenHelper** class we can easily create required database and tables for our application
- To use **SQLiteOpenHelper**, override the **onCreate()** and **onUpgrade()** methods (See the example)

Insert data into SQLite Database

- we can insert data into SQLite database by passing **ContentValues** to **insert()** method

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
//Create a new map of values, where column names are the keys
ContentValues cValues = new ContentValues();
cValues.put(COLUMN_NAME, name);
cValues.put(COLUMN_EMAIL, email);
// Insert the new row, returning the primary key value of the new row
long newRowId = db.insert(TABLE_Users,null, cValues);
```

Retrieving data from SQLite Database

- we can read the data from SQLite database using **query()** method

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
Cursor cursor = db.query(TABLE_Users, new String[]{COLUMN_NAME, COLUMN_E
MAIL}, COLUMN_EMAIL+ "=?",new String[]{String.valueOf(userid)},null,null, null);
//tblName, columns, selection, selectionargs, groupBy, having, OrderBy
```

Update data from SQLite Database

- we can update the data in SQLite database using **update()** method in android applications

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
ContentValues cVals = new ContentValues();
cVals.put(COLUMN_NAME, name);
int count = db.update(TABLE_Users, cVals, COLUMN_EMAIL+" = ?",new String[]{Str
ing.valueOf(id)});
```

Deleting data from SQLite Database

- we can delete data from SQLite database using **delete()** method

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
db.delete(TABLE_Users, COLUMN_EMAIL+" = ?",new String[]{String.valueOf(userid)});
```

Web Service with PHP/Java & MySQL

- External Databases are also the data repository for android apps for example apps that provide weather information, exchange rates, world news etc
- MySQL is one of the database tool. Connection between an android app and mysql can be done through
  - Web Services
  - JDBC without web services

Web Services

- A **Web Service is a Consumer_Machine-to-Provider_Machine** collaboration schema that operates over a computer network.
- The data exchanges occur independently of the ***OS, browser, platform, and programming languages*** used by the **provider** and the **client**.

- Web Services expect the computer network to support standard Web protocols such as XML, HTTP, HTTPS, FTP, and SMTP.

Advantages of Using the Web Service Architecture

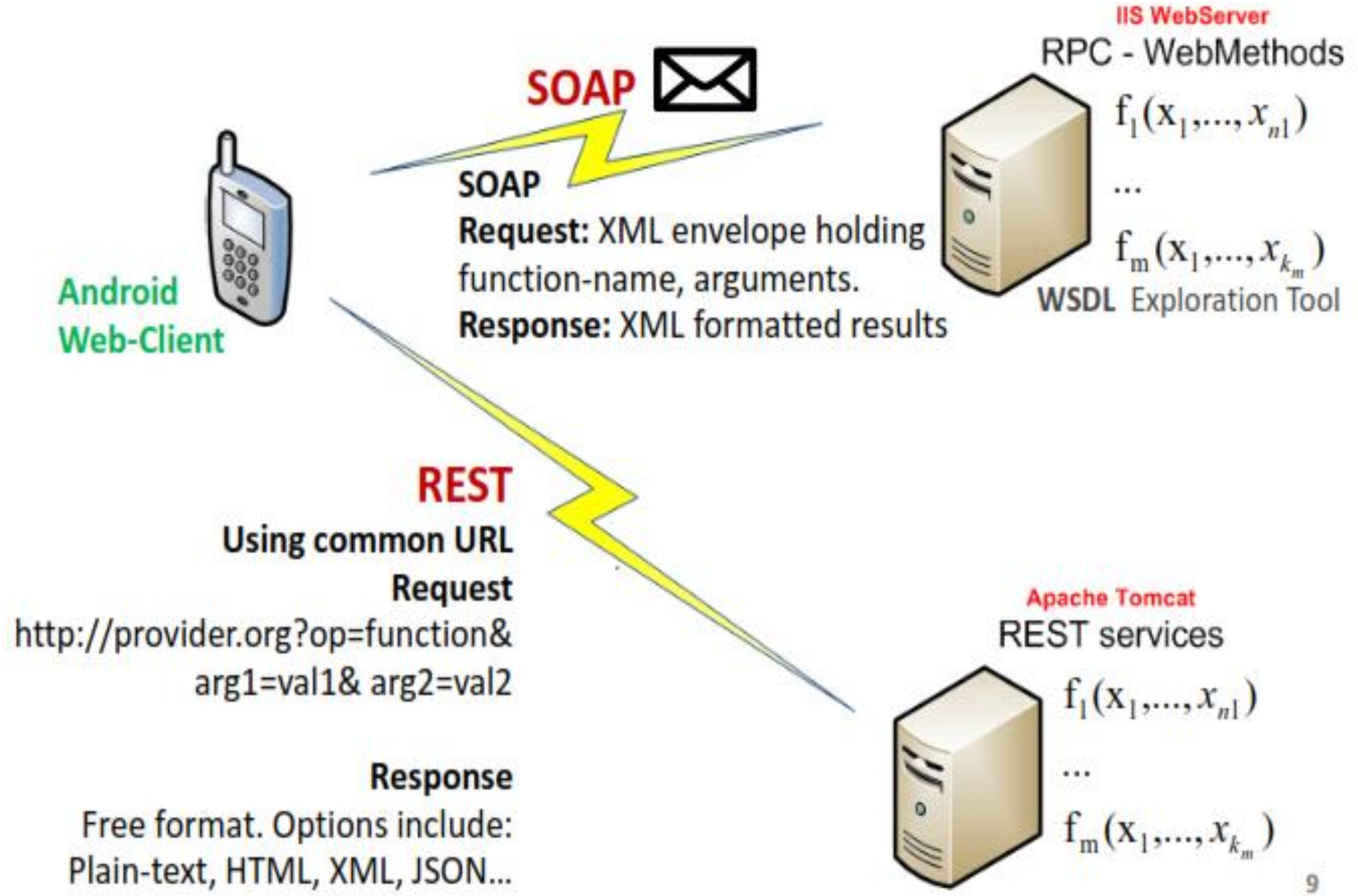
- **invoked functions are *implemented once (in the server) and called many times (by the remote users)*.**
- Elimination of redundant code,
- Ability to transparently make changes on the server to update a particular service function without clients having to be informed.
- Reduced maintenance and production costs.

There are two widely used forms of invoking and using Web Services

- **Representational State Transfer (REST)**
  - Closely tie to the **HTTP protocol** by associating its operation to the common methods: GET, POST, PUT, DELETE for HTTP/HTTPS.
  - has a **simple invocation mode and little overhead**.
  - Service calls rely on a **URL** which may also carry **arguments**. Sender & receiver must have an understanding of how they pass data items from one another. As an example: **Google Maps API uses the REST model**.
- **Remote Procedure Call (RPC).**
  - Remote services are seen as coherent collections of discoverable functions (or method calls) stored and exposed by EndPoint providers.
  - Some implementations of this category include: Simple Object Access Protocol (SOAP), Common Object Request Broker Architecture (CORBA), Microsoft's Distributed Component Object Model (DCOM) and Sun Microsystems's Java/Remote Method Invocation (RMI).

REST Vs SOAP

- **REST users refer to their remote services through a conventional URL** that commonly includes the location of the (stateless) **server**, the **service name**, the **function to be executed** and the **parameters needed by the function to operate** (if any). Data is transported using HTTP/HTTPS.
- **SOAP requires some scripting effort to create an XML envelop in which** data travels. An additional **overhead on the receiving end is needed to extract data from** the XML **envelope**. SOAP accepts a variety of transport mechanisms, among them HTTP, HTTPS, FTP, SMTP, etc.
- SOAP uses **WSDL (Web Service Description Language) for exposing** the format and operation of the services.



SOAP

Request: XML envelope holding function-name, arguments.

Response: XML formatted results

REST

Using common URL

Request

http://provider.org?op=function& arg1=val1& arg2=val2

Response

Free format. Options include: Plain-text, HTML, XML, JSON...

Connecting Android App with MySQL using RESTful web service

- **Download and install** XAMPP(or WAMP)
- Check whether the server work properly or not
  - Create a folder called test-android under C:\xampp\htdocs\ folder
  - Test the server by running http://localhost/test-android/
- Open the phpMyAdmin by visiting http://localhost/phpMyAdmin and create a database and table for your app
- Create connection and php files to handle database operation
- Create android project, Create the activities and create **JSONParser** for data exchange format
- Add the following to the application build gradle

android{ useLibrary '**org.apache.http.legacy**' }

dependencies{ compile "**org.apache.httpcomponents:httpcore:4.3.2**" }

- Allow remote connection permission in AndroidManifest.xml file.

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

- Use http://10.0.2.2/app-folder/phpfile or use an IP address if you want to access the database from your device