EAST WEST UNIVERSITY Department of Computer Science & Engineering

Android Storage

Data Storage In Android

- App State Storage:- Store app state data in memory
- Shared Preferences:- Store private primitive data in key-value pairs.
- Internal Storage:- Store private data on the device memory (it’s not RAM).
- External Storage:- Store public data on the shared external storage.
- SQLite Databases:- Store structured data in a private database.
- Network Connection:- Store data on the web with your own network server.

Activity State

- Activity’s state information can be lost, if it’s closed
  - When activity is no longer on the screen or if it is closed **because of freeing memory**
  - When **screen rotation** is changed, the activity is destroyed and opened again

How to Store State Information

- Store state:
  - onSaveInstanceState(Bundle)
- Read state
  - onRestoreInstanceState(Bundle)
- This will store data only temporarily: **for app lifetime!**
- Data will be held **in memory until the app is closed!**

Temporarily Store App State Information in Memory

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    String text = tv.getText().toString();
    savedInstanceState.putString("someKey", text);
    super.onSaveInstanceState(savedInstanceState);
}
```

Load Previously Stored App State Information from Memory

```
 @Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {

   super.onRestoreInstanceState(savedInstanceState);

   if (savedInstanceState != null) {

         String strValue = savedInstanceState.getString("someKey");

         if (strValue != null) {
             textfield.setText(strValue);
         }
     }
}
```

Shared Preferences

- Useful for **storing and retrieving** **primitive** **data** in (**key, value) pairs**

- **Lightweight usage**, such as saving application settings

- Typical usage of SharedPreferences is for **saving** application data such as **username and password, auto login flag, remember-user flag** etc.

Shared Preferences: How does it store internally

- The shared preferences **information is stored** in an **XML** file on the device
  - Typically in /data/data/<Your Application's packagename>/shared_prefs

- SharedPreferences can be associated with the entire application, or to a specific activity.

Shared Preferences: Declaration & Creation

- Example:
  - SharedPrefernces prefs = this.getSharedPrefernces("myPrefs", MODE_PRIVATE)
- If the preferences XML file exist, it is opened, otherwise it is created.
- To Control **access permission to the file**:
  - MODE_PRIVATE: private only to the application, all activities of that app can access
  - MODE_WORLD_READABLE: all application can read XML file
  - MODE_WORLD_WRITABLE: all application can write XML file

SharedPrefernces prefs = this.**getPrefernces**()

Shared Preferences: Storing Data

- To add Shared preferences, first an editor object is needed

```
SharedPrefernces prefs = this.getSharedPrefernces("myPrefs",  MODE_PRIVATE)
Editor prefsEditor = prefs.edit();
```

- Then, use the put() method to add the key-value pairs

  - prefsEditor.putString("username", "D-Link");
  - prefsEditor.putString("password", "vlsi#1@2");
  - prefsEditor.putint("times-login", 1);
  - ~~prefsEditor.commit()~~;
  - prefsEditor.apply();

Shared Preferences: Retrieving Previously Stored Data

- To retrieve shared preferences data:

```
SharedPrefernces prefs = this.getSharedPrefernces("myPrefs",  MODE_PRIVATE)
String username = prefs.getString("username", "");
String password = prefs.getString("password", "");
```

Shared Preferences: Private to a specific activity

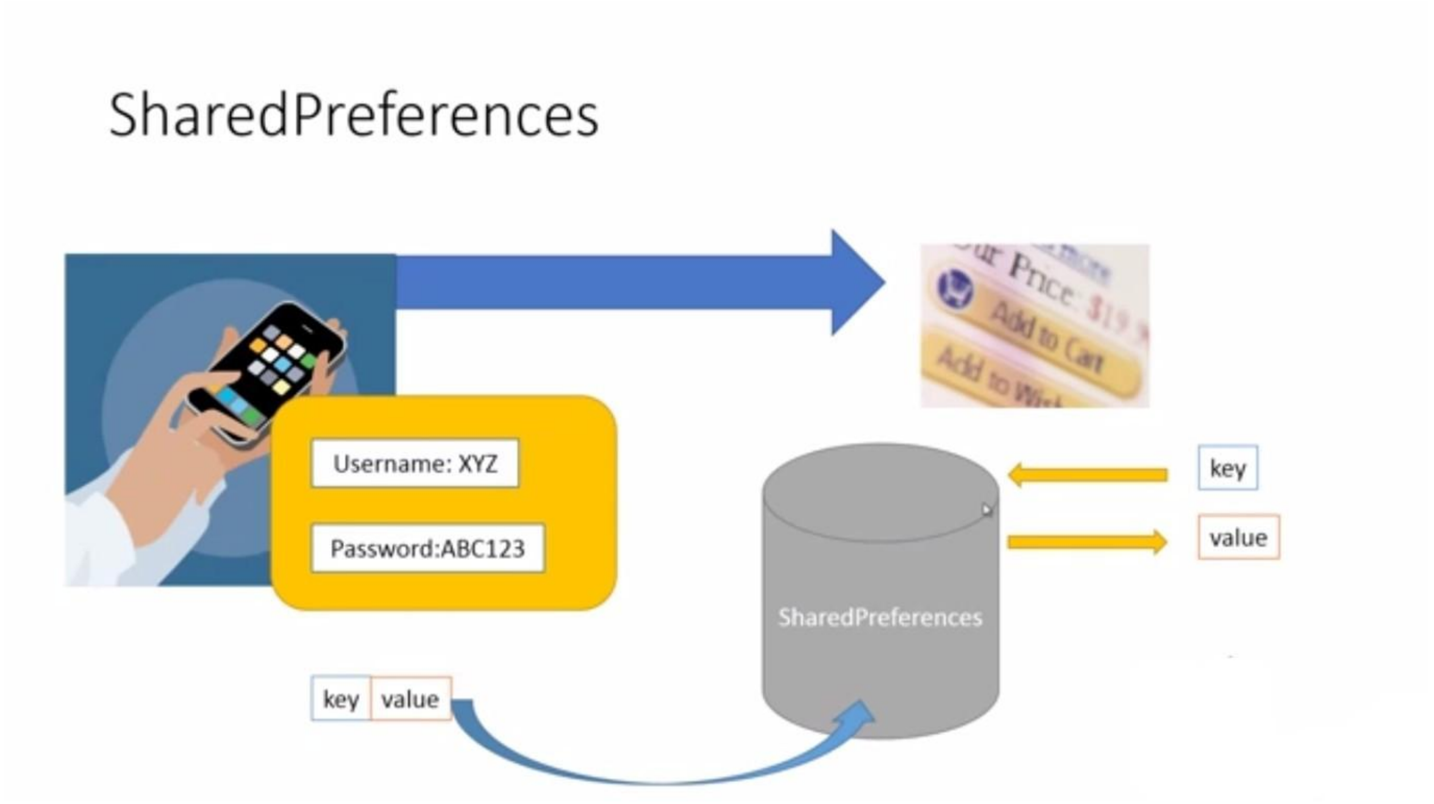
- If you are using SharedPrefernces for **specific activity**, then use **getPreferences()** method
  - No need to specify the name of the preferences file

```
SharedPrefernces prefs = this.getPrefernces()
Editor prefsEditor = prefs.edit();
prefsEditor.putString("username", "D-Link");
prefsEditor.putString("password", "vlsi#1@2");
prefsEditor.putint("times-login", 1);
prefsEditor.apply();
```

- Retrieval of data

```
String username = prefs.getString("username", "");
String password = prefs.getString("password", "");
```

Overview of SharedPreferences



Visit the following online contents to learn more about SharedPreferences

https://www.geeksforgeeks.org/shared-preferences-in-android-with-examples/

Internal Storage

- Android can save files **directly to the device internal storage**.

- These files are **private to the application** and will be removed if you uninstall the application.

- We can create a file using **openFileOutput()** with parameter as file name and the operating mode.

- Generally not recommended to use flat files for storing text data.

Internal Storage Contd….

- Similarly, we can open the file using **openFileInput()** passing the parameter as the **filename with extension**.
- File are used to **store large amount of data**
- Use **I/O interfaces** provided by **java.io.*** libraries to read/write files.
- **Only local files** can be accessed.

File Operation(Read)

- Use **context.openFileInput(String name)** to open a private input file stream related to a program.

- Throw **FileNotFoundException** when file does not exist.

- Syntax: FileInputStream in = this.openFileInput("xyz.txt")

.
.
.
.
.

in.close()://Close input stream

File Operation (Write)

- Use **context.openFileOutput(String name, int mode)** to open a private output file stream related to a program.

- The file will be created if it does not exist.

- Output stream can be opened in append mode, which means new data will be appended to end of the file.

File Operation (write) : Example

- Syntax:-

String myString = "Hello World"

//Open and Write in”myfile.txt”, using append mode.

FileOutputStream outfile = this.openFileOutput("myfile.txt", MODE_APPEND)

outfile.write(myString.getBytes());

outfile.close(); //close output stream

What will happen if not closed?

External Storage

- Every Android-compatible device supports a shared “**external storage**” that you can use to save files
  - Removable storage media (such as an SD card)
  - Internal (non-removable) storage

- File saved to the external storage are world readable and can be modified by the user when they enable USB mass storage to transfer files on computer.

External Storage Continue

- Must check whether external storage is available first by calling **getExternalStorageState()** **(why?)**
  - Also check whether it allows read/write before reading/writing on it

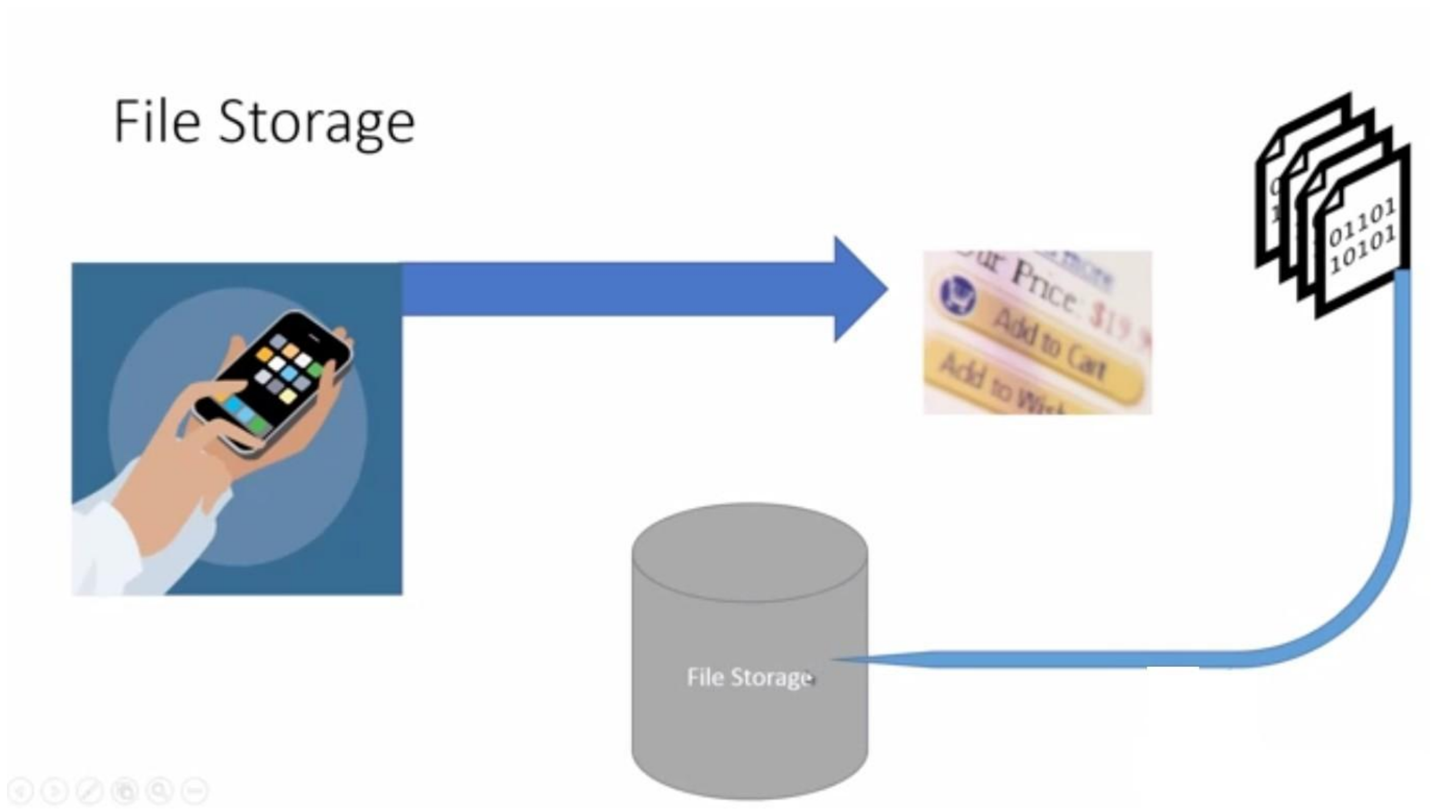
- getExternalFilesDir() takes a parameter such as DIRECTORY_MUSIC, DIRECTORY_RINGTONE etc, to open specific type of subdirectories.

- For **public** shared directories, use **getExternalStoragePublicDirectory()**

External Storage Contd…..

- For **cache files**, use **getExternalCacheDir()**
- All these are applicable for **API level 8 or above**
- For API level 7 or below ,use the method;
  - **getExternalStorageDirectory()**
  - Private files stored in//Android/data/<package_name>/files/
  - Cache files stored in//Android/data/<package_name>/cache/

Example


File Storage
Our Price $19
Add to Cart
Add to Wish
01101
10101
File Storage

Static files

- You can save static files into res/raw directory

- Accessing using openRawResource(R.raw.<filename>)
- Returns InputStream

- Cannot write to data
- Why?

SQLite Databases

- **android.database.sqlite** Contains the SQLite database management classes that an application would use to manage its own private database.

- **android.database.sqlite.SQLiteDatabase** Contains the methods for: creating, opening, closing, inserting, updating, deleting and quering an SQLite database.

android.database.sqlite - Classes

- **SQLiteCloseable** - An object created from a SQLiteDatabase that can be closed.
- **SQLiteCursor** - A Cursor implementation that exposes results from a query on a SQLiteDatabase.
- **SQLiteDatabase** - Exposes methods to manage a SQLite database.
- **SQLiteOpenHelper** - A helper class to manage database creation and version management.
- **SQLiteProgram** - A base class for compiled SQLite programs.
- **SQLiteQuery** - A SQLite program that represents a query that reads the resulting rows into a CursorWindow.
- **SQLiteQueryBuilder** - a convenience class that helps build SQL queries to be sent to SQLiteDatabase objects.
- **SQLiteStatement** - A pre-compiled statement against a SQLiteDatabase that can be reused.

OpenOrCreateDatabase

- This method will open an existing database or create one in the application data area

```
import android.database.sqlite.SQLiteDatabase;

SQLiteDatabase myDatabase;
myDatabase = openOrCreateDatabase ("my_sqlite_database.db",
                    SQLiteDatabase.CREATE_IF_NECESSARY ,  null);
```

If there exists a database file from where database should be imported

Creating Tables

- Create a static string containing the SQLite CREATE statement, use the execSQL( ) method to execute it.

```
String  createAuthor = "CREAT TABLE      authors (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    fname TEXT,  lname  TEXT);

myDatabase.execSQL(creat Author);
```

Supported Data Types

Not all data types are supported

Type	Meaning
NULL	The null value
INTEGER	Any number which is no floating point number
REAL	Floating-point numbers (8-Byte IEEE 754 - i.e. double precision)
TEXT	Any String and also single characters (UTF-8, UTF-16BE or UTF-16LE)
BLOB	A binary blob of data

insert( )

- long insert(String table, String nullColumnHack, ContentValues values)

```
import android.content.ContentValues;

ContentValues values = new ContentValues( );
values.put("firstname" , "J.K.");
values.put("lastname" , "Rowling");
long newAuthorID = myDatabase.insert("tbl_authors", null ,  values);
```

Alternatives:

```
String q = “INSERT INTO tbl_authors("firstname" , "lastname")  VALUES("J.K. ", "Rowling")”;
myDatabase.execSQL(q);
```

Update()

- int update(String table, ContentValues values, String whereClause, String[ ] whereArgs)

```
public void updateBookTitle(Integer pubId, String authName, String newTitle) {

  ContentValues values = new ContentValues();
  values.put("title" , newTitle);

  myDatabase.update("tbl_books", values, ”pub_id=?,auth_name=?", new String[ ] {pubId, authName} );
}
```

Alternatives:

```
String q = “UPDATE tbl_authors SET newTitle="title" WHERE pub_id=pubId AND auth_name= authName”;
myDatabase.execSQL(q);
```

Execute SQL Query

```
int delete(String table, String whereClause, String[]  whereArgs)
```

```
public void deleteBook(Integer bookId) {
		myDatabase.delete("tbl_books" , "id=?", new String[ ] {bookId.toString()}) ;
}
```

Alternatives:

```
db.execSQL(“DELETE FROM table_name WHERE value like ‘value%’”)
```

Use of Helper Class

Create/open database

Executed only once in app life time

Executed whenever DB-version is updated

Get database reference before query excution

Don’t forget to close database

DB Version

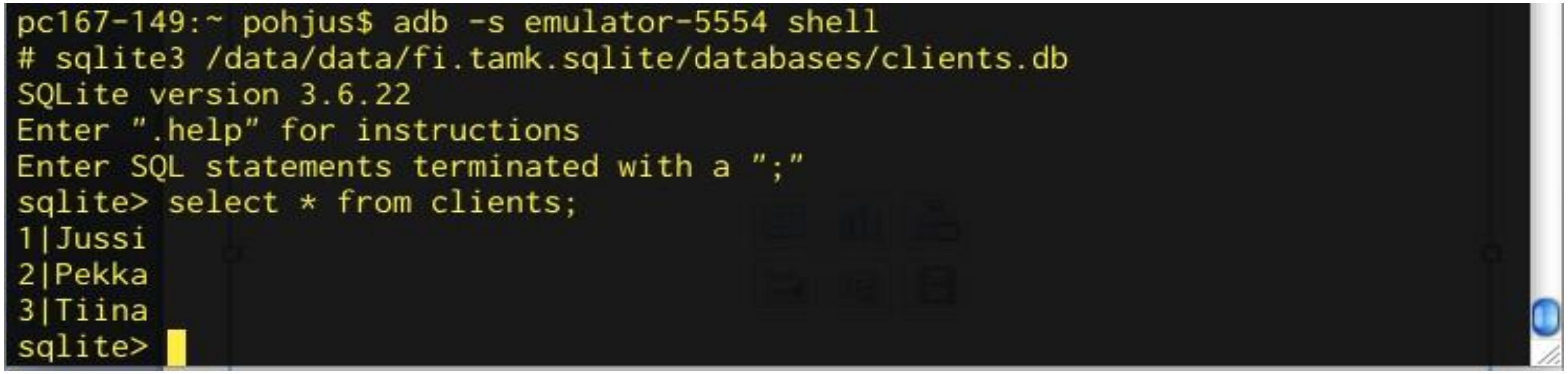
```
public class KeyValueDB extends SQLiteOpenHelper {

    public KeyValueDB(Context context) {
        super(context, "MyDBName.db", null, 1);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        System.out.println("DB@OnCreate");
        db.execSQL("create table my_table  (col1 TEXT,  col2 TEXT)");
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("ALTER table my_table  ......");
        db.execSQL("ALTER table my_table2  ......");
        db.execSQL("ALTER table my_table3 ......");
    }
    public void insertQuery(....) {
        SQLiteDatabase db = this.getWritableDatabase();
        ...
    }
    public void updateQuery(....) {
        SQLiteDatabase db = this.getWritableDatabase();
        ...
    }
    public Cursor execute(String query) {
        SQLiteDatabase db = this.getWritableDatabase();
        Cursor res;
        try {
            res = db.rawQuery(query, null);
        } catch (Exception e){
            e.printStackTrace();
        }
        return res;
    }
    .....
}
```

SQLite

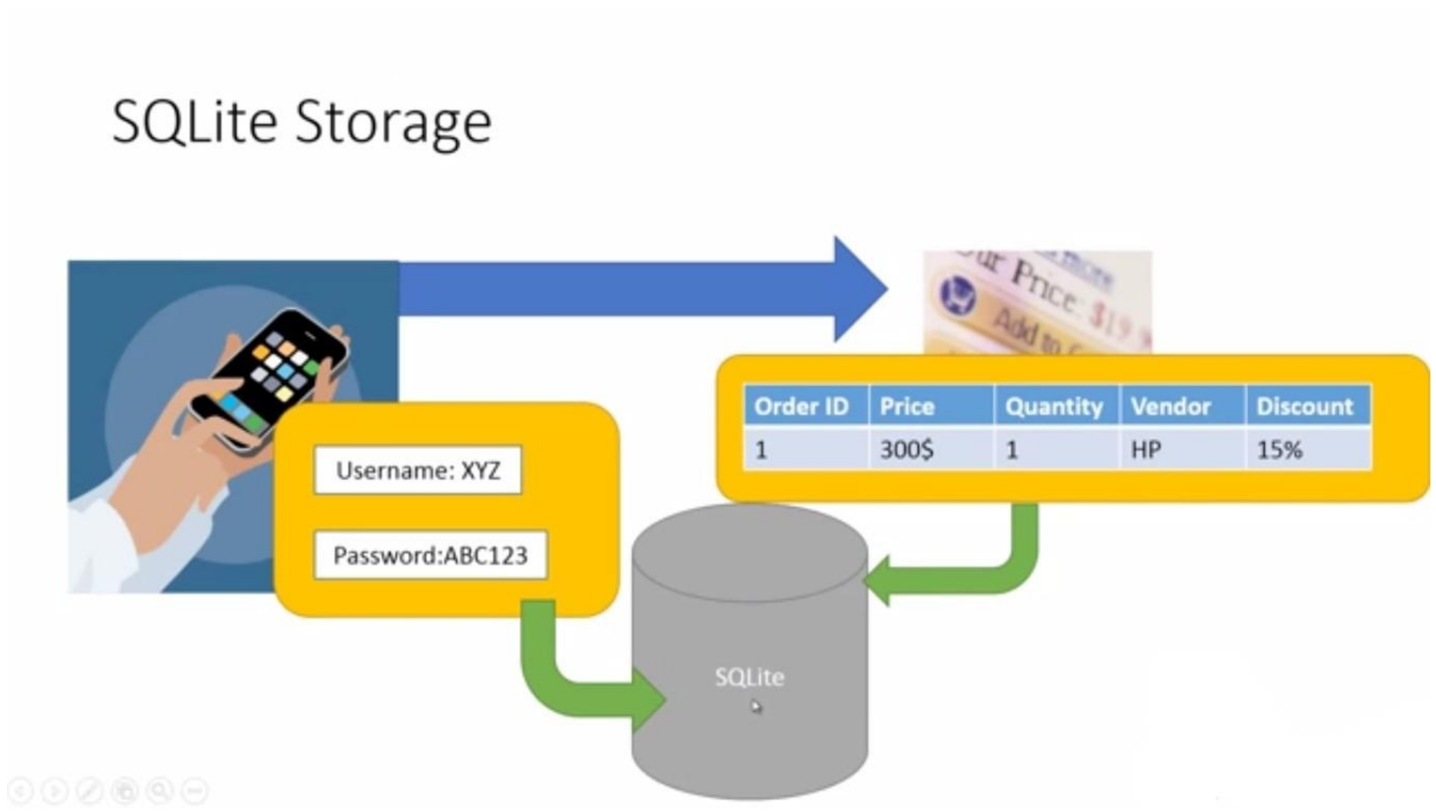
- Android SDK has a tool called sqlite3 which enables you to browse table contents using sql commands and command line

- All databases are stored in /data/data/ <package_name>/databases folder on your device.



```
pc167-149:~ pohjus$ adb -s emulator-5554 shell
# sqlite3 /data/data/fi.tamk.sqlite/databases/clients.db
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from clients;
1|Jussi
2|Pekka
3|Tiina
sqlite>
```

SQLite Storage


SQLite Storage

Username: XYZ

Password:ABC123

Order ID	Price	Quantity	Vendor	Discount
1	300$	1	HP	15%

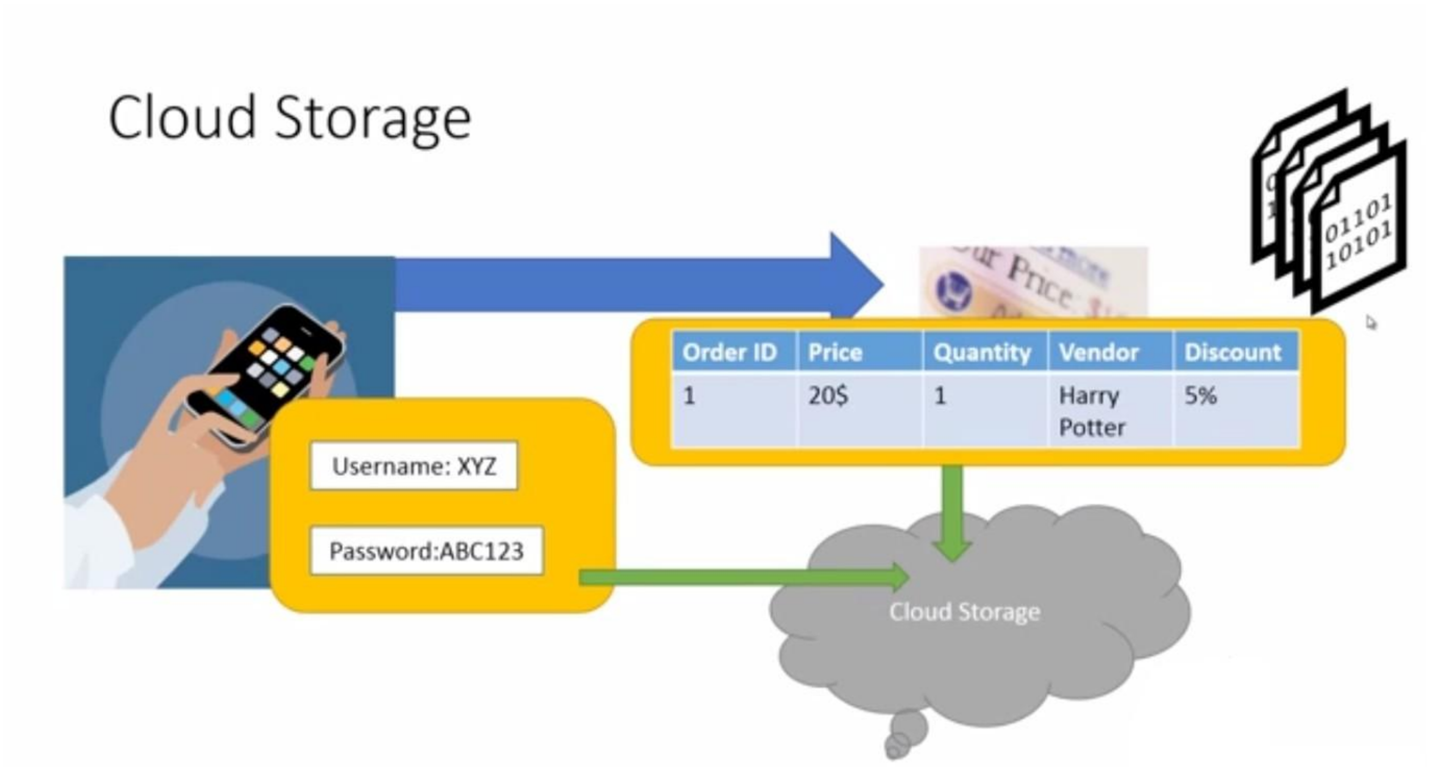
Cloud Storage

- Online file storage centres or cloud storage providers allow you to safely upload your files to the Internet.

Cloud Storage Contd.....

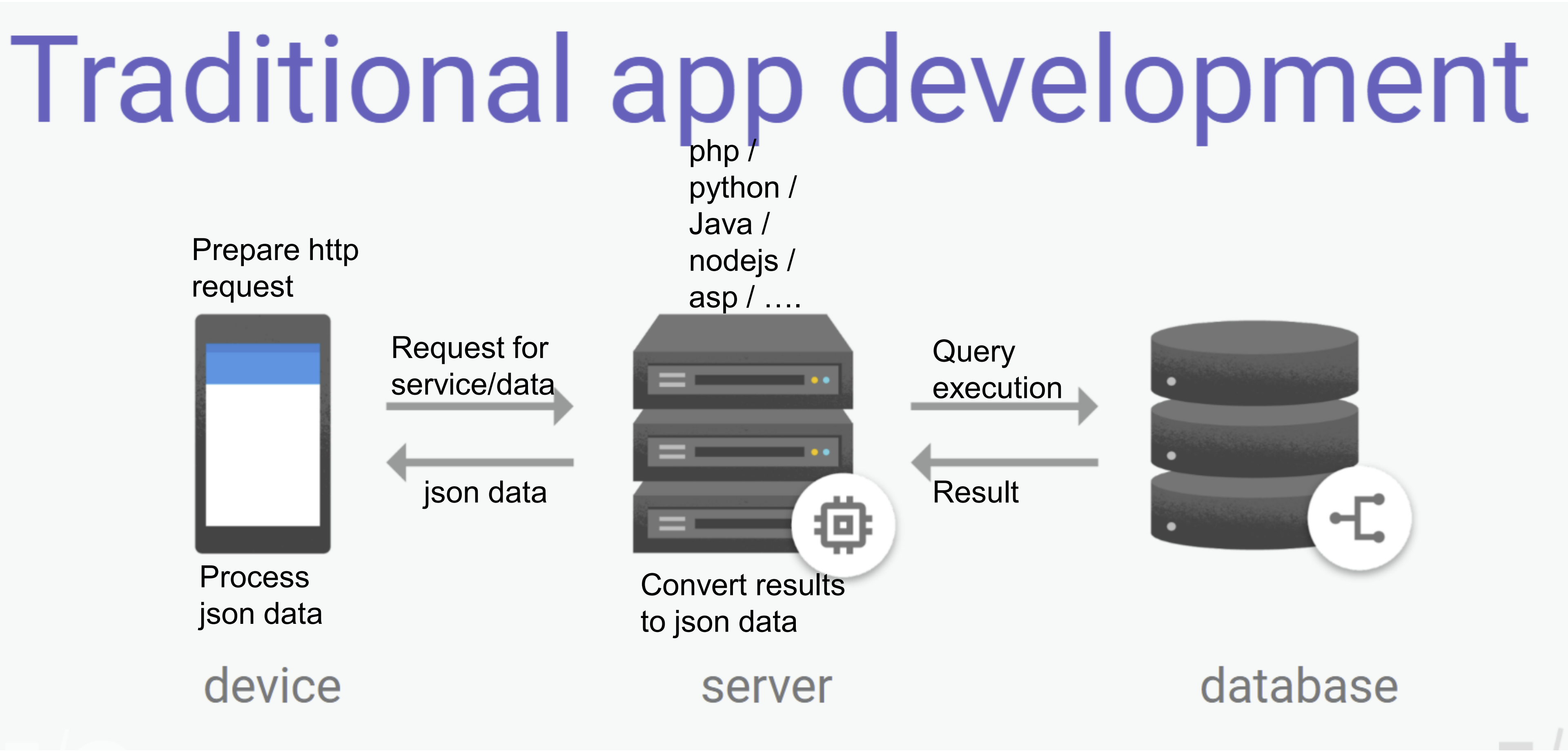
- There are various providers of cloud storage
- Examples:
  - Apple iCloud(Gives 5GB of free storage)
  - Dropbox(Gives 2GB of free storage)
  - Google Drive(Gives 15GB of free storage)
  - Amazon Cloud Drive(Gives 5GB of free storage
  - Microsoft SkyDrive(Gives 7GB of free storage)

Example

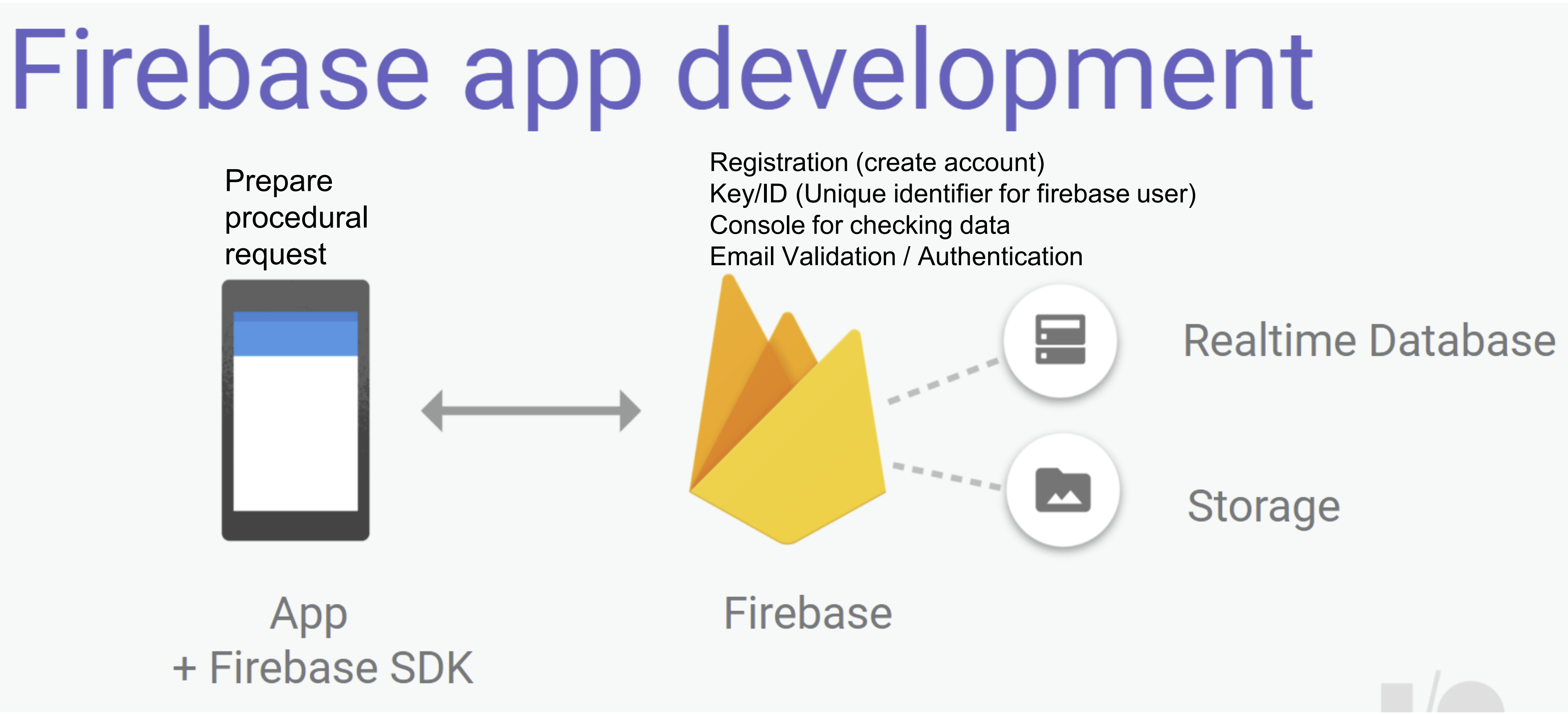


Order ID	Price	Quantity	Vendor	Discount
1	20$	1	Harry Potter	5%

Remote Database



Firebase Remote Database


Firebase app development
Prepare procedural request
Registration (create account)
Key/ID (Unique identifier for firebase user)
Console for checking data
Email Validation / Authentication
Realtime Database
Storage
App + Firebase SDK
Firebase

Simple Implementation

```
// Write a message to the database
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");

myRef.setValue("Hello, World!");
```

```
// Read from the database
myRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        // This method is called once with the initial value and again
        // whenever data at this location is updated.
        String value = dataSnapshot.getValue(String.class);
        Log.d(TAG, "Value is: " + value);
    }

    @Override
    public void onCancelled(DatabaseError error) {
        // Failed to read value
        Log.w(TAG, "Failed to read value.", error.toException());
    }
});
```

Extra Slides

SQLiteOpenHelper

android.database.sqlite.SQLiteOpenHelper

- It is a helper class to manage database creation and version management.

Public Constructors	
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)	Create a helper object to create, open, and/or manage a database.
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)	Create a helper object to create, open, and/or manage a database.

Public Methods		
synchronized void close()		Close any open database object.
String	getDatabaseName()	Return the name of the SQLite database being opened, as given to the constructor.
SQLiteDatabase	getReadableDatabase()	Create and/or open a database.
SQLiteDatabase	getWritableDatabase()	Create and/or open a database that will be used for reading and writing.
abstract void	onCreate(SQLiteDatabase db)	Called when the database is created for the first time.
void	onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)	Called when the database needs to be downgraded.
void	onOpen(SQLiteDatabase db)	Called when the database has been opened.
abstract void	onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	Called when the database needs to be upgraded.

SQLiteDatabase

android.database.sqlite.SQLiteDatabase

- Database names must be unique within an application, not across all applications.

Public Methods		
static SQLiteDatabase	create(SQLiteDatabase.CursorFactory factory)	Create a memory backed SQLite database.
int	delete(String table, String whereClause, String[] whereArgs)	Convenience method for deleting rows in the database.
static boolean	deleteDatabase(File file)	Deletes a database including its journal file and other auxiliary files that may have been created by the database engine.
void	execSQL(String sql)	Execute a single SQL statement that is NOT a SELECT or any other SQL statement that returns data.
void	execSQL(String sql, Object[] bindArgs)	Execute a single SQL statement that is NOT a SELECT/INSERT/UPDATE/DELETE.
long	getMaximumSize()	Returns the maximum size the database may grow to.
final String	getPath()	Gets the path to the database file.
int	getVersion()	Gets the database version.
long	insert(String table, String nullColumnHack, ContentValues values)	Convenience method for inserting a row into the database.
boolean	isOpen()	Returns true if the database is currently open.
boolean	isReadOnly()	Returns true if the database is opened as read only.
static SQLiteDatabase	openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)	Open the database according to the flags OPEN_READWRITE OPEN_READONLY CREATE_IF_NECESSARY and/or NO_LOCALIZED_COLLATORS.

SQLiteDatabase

android.database.sqlite.SQLiteDatabase

Public Methods		
static SQLiteDatabase	openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)	Open the database according to the flags OPEN_READWRITE OPEN_READONLY CREATE_IF_NECESSARY and/or NO_LOCALIZED_COLLATORS.
static SQLiteDatabase	openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory, DatabaseErrorHandler errorHandler)	Equivalent to openDatabase(path, factory, CREATE_IF_NECESSARY, errorHandler).
static SQLiteDatabase	openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)	Equivalent to openDatabase(path, factory, CREATE_IF_NECESSARY).
static SQLiteDatabase	openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)	Equivalent to openDatabase(file.getPath(), factory, CREATE_IF_NECESSARY).
Cursor	query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)	Query the given table, returning a Cursor over the result set.
Cursor	rawQuery(String sql, String[] selectionArgs, CancellationSignal cancellationSignal)	Runs the provided SQL and returns a Cursor over the result set.
Cursor	rawQuery(String sql, String[] selectionArgs)	Runs the provided SQL and returns a Cursor over the result set.
long	setMaximumSize(long numBytes)	Sets the maximum size the database will grow to.
void	setVersion(int version)	Sets the database version.
String	toString()	Returns a string containing a concise, human-readable description of this object.
int	update(String table, ContentValues values, String whereClause, String[] whereArgs)	Convenience method for updating rows in the database.

ContentValues

android.content.ContentValues

- This class is used to store a set of values.

Public Constructors	
ContentValues()	Creates an empty set of values using the default initial size
ContentValues(int size)	Creates an empty set of values using the given initial size
ContentValues(ContentValues from)	Creates a set of values copied from the given set

Public Methods		
void	clear()	Removes all values.
boolean	containsKey(String key)	Returns true if this object has the named value.
boolean	equals(Object object)	Compares this instance with the specified object and indicates if they are equal.
Object	get(String key)	Gets a value.
Boolean	getAsBoolean(String key)	Gets a value and converts it to a Boolean.
Byte	getAsByte(String key)	Gets a value and converts it to a Byte.
byte[]	getAsByteArray(String key)	Gets a value that is a byte array.
Double	getAsDouble(String key)	Gets a value and converts it to a Double.
Float	getAsFloat(String key)	Gets a value and converts it to a Float.
Integer	getAsInteger(String key)	Gets a value and converts it to an Integer.
Long	getAsLong(String key)	Gets a value and converts it to a Long.
Short	getAsShort(String key)	Gets a value and converts it to a Short.
String	getAsString(String key)	Gets a value and converts it to a String.
void	put(String key, Byte value)	Adds a value to the set.
void	put(String key, Integer value)	Adds a value to the set.

ContentValues

Public Methods		
void	put(String key, Float value)	Adds a value to the set.
void	put(String key, Short value)	Adds a value to the set.
void	put(String key, byte[] value)	Adds a value to the set.
void	put(String key, String value)	Adds a value to the set.
void	put(String key, Double value)	Adds a value to the set.
void	put(String key, Long value)	Adds a value to the set.
void	put(String key, Boolean value)	Adds a value to the set.
void	putAll(ContentValues other)	Adds all values from the passed in ContentValues.
void	putNull(String key)	Adds a null value to the set.
void	remove(String key)	Remove a single value.
int	size()	Returns the number of values.
String	toString()	Returns a string containing a concise, human-readable description of this object.

Cursor

android.database.Cursor

- This interface provides random read-write access to the result set returned by a database query.

Public Methods		
abstract void	close()	Closes the Cursor, releasing all of its resources and making it completely invalid.
abstract void	copyStringToBuffer(int columnIndex, CharArrayBuffer buffer)	Retrieves the requested column text and stores it in the buffer provided.
abstract int	getColumnCount()	Return total number of columns
abstract int	getColumnIndex(String columnName)	Returns the zero-based index for the given column name, or -1 if the column doesn't exist.
abstract int	getColumnIndexOrThrow(String columnName)	Returns the zero-based index for the given column name, or throws IllegalArgumentException if the column doesn't exist.
abstract String	getColumnName(int columnIndex)	Returns the column name at the given zero-based column index.
abstract String[]	getColumnNames()	Returns a string array holding the names of all of the columns in the result set in the order in which they were listed in the result.
abstract int	getCount()	Returns the numbers of rows in the cursor.
abstract double	getDouble(int columnIndex)	Returns the value of the requested column as a double.
abstract Bundle	getExtras()	Returns a bundle of extra values.
abstract float	getFloat(int columnIndex)	Returns the value of the requested column as a float.
abstract int	getInt(int columnIndex)	Returns the value of the requested column as an int.
abstract long	getLong(int columnIndex)	Returns the value of the requested column as a long.
abstract int	getPosition()	Returns the current position of the cursor in the row set.
abstract short	getShort(int columnIndex)	Returns the value of the requested column as a short.
abstract String	getString(int columnIndex)	Returns the value of the requested column as a String.

Cursor

Public Methods		
abstract int	getType(int columnIndex)	Returns data type of the given column's value.
abstract boolean	isAfterLast()	Returns whether the cursor is pointing to the position after the last row.
abstract boolean	isBeforeFirst()	Returns whether the cursor is pointing to the position before the first row.
abstract boolean	isClosed()	return true if the cursor is closed
abstract boolean	isFirst()	Returns whether the cursor is pointing to the first row.
abstract boolean	isLast()	Returns whether the cursor is pointing to the last row.
abstract boolean	isNull(int columnIndex)	Returns true if the value in the indicated column is null.
abstract boolean	move(int offset)	Move the cursor by a relative amount, forward or backward, from the current position.
abstract boolean	moveToFirst()	Move the cursor to the first row.
abstract boolean	moveToLast()	Move the cursor to the last row.
abstract boolean	moveToNext()	Move the cursor to the next row.
abstract boolean	moveToPosition(int position)	Move the cursor to an absolute position.
abstract boolean	moveToPrevious()	Move the cursor to the previous row.