Android: Data Storage

Content based on Android Storage Options documentation

Data Storage Options

Shared Preferences
Internal Storage
External Storage
SQLite Databases
Network Connection

Shared Preferences

Shared Preferences

- Lightweight mechanism for storing key/value data
  - Key type is a String
  - Value type is Integer, Long, Double, Boolean, or String
- Great option storing primitive data
- If you have a relatively small collection of key values that you'd like to save, you should use the `SharedPreferences` APIs.
- A `SharedPreferences` object points to a file containing key-value pairs and provides simple methods to read and write them.

Shared Preferences

`getPreferences()` - Use this if you need only one preferences file for your Activity.

`getSharedPreferences()` - Use this if you need multiple preferences files identified by name.

Access Modes

`MODE_PRIVATE` - private to app (recommended)
`MODE_WORLD_READABLE` - any app can read
`MODE_WORLD_WRITEABLE` - any app can write

Writing to SharedPreferences

- After obtaining SharedPreferences object:
  - call edit() method on object to get a SharedPreferences.Editor object
  - place data by calling put methods on the SharedPreferences.Editor object
- When done writing data via the editor call either apply() or commit()
- apply() is the simpler method
  - used when only one process expected to write to the preferences object
- commit() returns a boolean if write was successful
  - for when multiple process may be writing to preferences

Shared Preferences - Example Writing

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);

SharedPreferences.Editor editor = sharedPref.edit();

editor.putInt(getString(R.string.saved_high_score), newHighScore);

editor.commit();
```

Reading From Shared Preferences

After obtaining SharedPreferences object use various get methods to retrieve data

Provide key (string) and default value if key is not present

get Boolean, Float, Int, Long, String, StringSet

getAll() returns Map<String, ?> with all of the key/value pairs in the preferences

Shared Preferences - Example Reading

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);

int defaultValue =
getResources().getInteger(R.string.saved_high_score_default);

long highScore =
sharedPref.getInt(getString(R.string.saved_high_score),defaultValue);
```

Shared Preferences - Location

Data is stored under

`/data/data/$PACKAGE_NAME/shared_prefs`

Reference

https://developer.android.com/training/basics/data-storage/shared-preferences.html

Shared Preferences File

Stored as XML

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="victory_message">Excellent</string>
<int name="board_color" value="-65528" />
<int name="mTies" value="6" />
<string name="difficulty_level">Harder</string>
<int name="mComputerWins" value="1" />
<int name="mDifficulty" value="1" />
<int name="mHumanWins" value="9" />
</map>
```

Preference Activity

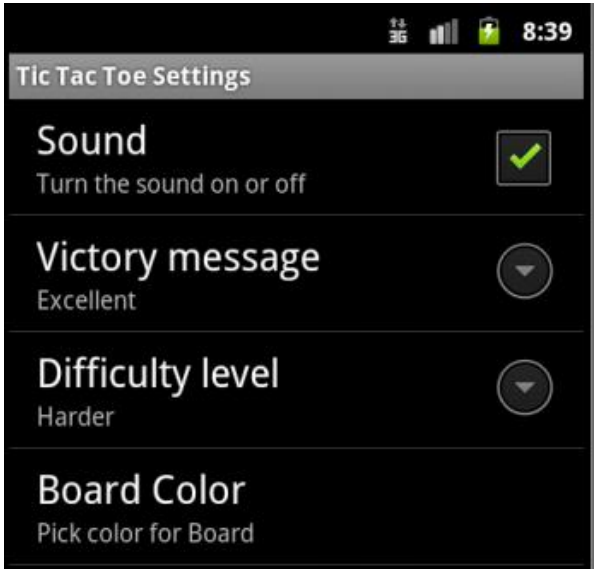
An Activity framework to allow user to select and set preferences for your app

an example:

difficulty, sound, color, victory message

Main Activity can start a preference activity to allow user to set preferences

Current standard is to use a PreferenceFragment instead



Internal Storage

Internal Storage

Save files directly to device's internal storage.

Private data by default

- Other applications cannot access it (nor can the user)

Removed when app is uninstalled.

Using the Internal Storage APIs

Writing to Internal Storage

1. Call openFileOutput() with the filename and mode.
2. Write to the file with write().
3. Close the stream with close().

Reading from Internal Storage

1. Call openFileInput() and pass it the filename.
2. Read bytes from the file with read().
3. Then close the stream with close().

Internal Storage - Example Writing

```
String filename = "diary.txt";
String body = "My deepest secret is...";

FileOutputStream fos = openFileOutput(filename,
    Context.MODE_PRIVATE);
fos.write(body.getBytes());
fos.close();
```

Internal Storage - Example Reading

```
BufferedReader reader = new BufferedReader(new
    InputStreamReader(openFileInput(FILENAME)));
Log.d("COEN268", "From file: " + reader.readLine());
```

Internal Storage - Location

Data is stored under

`/data/data/$PACKAGE_NAME/files`

External Storage

External Storage

May be removable storage media (e.g., SD card) or internal (non-removable) storage.

Files saved to the external storage are world-readable!

Avoid leaking data. Case study: WhatsApp

Can be modified by the user when they enable USB mass storage to transfer files on a computer.

External Storage

New files acquired through your app should be saved to a "public" location

Example public directories: Music/, Pictures/, Ringtones/, etc.

Public directories

The Environment class has constants for common public directories:

Environment.DIRECTORY_ALARMS - audio files that should be in the list of alarms that the user can select
Environment.DIRECTORY_DCIM - traditional location for pictures and videos when mounting the device as a camera
Environment.DIRECTORY_DOWNLOADS
Environment.DIRECTORY_MOVIES
Environment.DIRECTORY_MUSIC
Environment.DIRECTORY_NOTIFICATIONS - audio files that should be in the list of notifications that the user can select
Environment.DIRECTORY_PICTURES
Environment.DIRECTORY_PODCASTS
Environment.DIRECTORY_RINGTONES

External Storage - Permission

Writing to external storage requires the android.permission.WRITE_EXTERNAL_STORAGE permission.

Add to AndroidManifest.xml file:

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


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

External Storage - Example Saving

```
/* Checks if external storage is available for read and write */

public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

External Storage - Example Saving

```
public File getAlbumStorageDir(String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new
File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

External Storage - Example Saving

```
// Example from Taking Photos Simply
(http://developer.android.com/training/camera/photobasics.html)

private File createImageFile() throws IOException {
    // Create an image file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    String imageFileName = "JPEG_" + timeStamp + "_";
    File storageDir = getAlbumStorageDir("Zoo");
    File image = File.createTempFile(
        imageFileName,  /* prefix */
        ".jpg",         /* suffix */
        storageDir      /* directory */
    );
    return image;
}
```

External Storage - Example Saving

```
private static final int RESULT_CODE = 0;
private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException ex) {
            // Error occurred while creating the File
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT,
                    Uri.fromFile(photoFile));
            startActivityForResult(takePictureIntent, RESULT_CODE);
        }
    }
}
```

External Storage Location

In emulator, external storage is located at
/storage/sdcard/Pictures

Query Free Space

you can find out whether sufficient space is available without causing an IOException by calling getFreeSpace() or getTotalSpace().
These methods provide the current available space and the total space in the storage volume, respectively.

Delete a File

The most straightforward way to delete a file is to have the opened file reference call delete() on itself.

```
myFile.delete();
```

If the file is saved on internal storage, you can also ask the Context to locate and delete a file by calling deleteFile():

```
myContext.deleteFile(fileName);
```

Reference

https://developer.android.com/training/basics/data-storage/files.html

SQLite

SQLite Overview

Most widely deployed SQL database engine in the world!

Implements most of the SQL-92 standard

Great option for storing structured data

Open-source

Standards-compliant

Lightweight (less than 400kb)

Zero-configuration

SQL and Databases

SQL is a language used to manipulate and manage information in a relational database management system (RDBMS)

SQL Commands:

CREATE TABLE - creates a new database table

ALTER TABLE - alters a database table

DROP TABLE - deletes a database table

CREATE INDEX - creates an index (search key)

DROP INDEX - deletes an index

SQL Commands

SELECT - get data from a database table

UPDATE - change data in a database table

DELETE - remove data from a database table

INSERT INTO - insert new data in a database table

Content Values and Cursors

ContentValue - represents a single table row as a key/value map.

Cursor - pointer to result set

SQLiteOpenHelper

Helper class for creating, opening, and upgrading databases.

Create a subclass SQLiteOpenHelper for your database instance

SQLiteDatabase - APIs for CRUD

Operation	API Method
Creation	SQLiteDatabase.insert()
Read	SQLiteDatabase.query()
Update	SQLiteDatabase.update()
Deletion	SQLiteDatabase.delete()

Example: Zoo Database

Example: create a Zoo table. Each row represents an animal and has 4 fields:

- ID - a unique identifier for each row
- Name
- Description
- File Path - for the image of the animal

Example Content in Zoo Table

_id	name	description	file_path
1	alpaca	Furry, four-legged animal	/sdcard/alpaca.jpg
2	monkey	Funny, likes to jump.	/sdcard/monkey.jpg
3	whale	Big sea creature	/sdcard/whale.jpg
...	...	...	...

Database Schema

A database schema defines the structure of a database.

You can create a table with the CREATE TABLE query:

```
CREATE TABLE Zoo (
    _id integer primary key autoincrement,
    name text,
    description text,
    file_path text);
```

ZooDbHelper.java

```
public class ZooDbHelper extends SQLiteOpenHelper {
        public static final String ID_COLUMN = "_id";
        public static final String NAME_COLUMN = "name";
        public static final String DESCRIPTION_COLUMN = "description";
        public static final String FILE_PATH_COLUMN = "filepath";

        public static final String DATABASE_TABLE = "Zoo";
        public static final int DATABASE_VERSION = 1;
        private static final String DATABASE_CREATE = String.format(
                        "CREATE TABLE %s (" +
                        "  %s integer primary key autoincrement, " +
                        "  %s text," +
                        "  %s text," +
                        "  %s text)",
                        DATABASE_TABLE, ID_COLUMN, NAME_COLUMN,
DESCRIPTION_COLUMN, FILE_PATH_COLUMN);
```

ZooDbHelper.java (Continued)

```
        public ZooDbHelper(Context context) {
                super(context, DATABASE_TABLE, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
                db.execSQL(DATABASE_CREATE);
        }

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

Inserting

```
SQLiteDatabase db = new ZooDbHelper(this).getWritableDatabase();
ContentValues newValues = new ContentValues();
newValues.put(ZooDbHelper.NAME_COLUMN, "alpaca");
newValues.put(ZooDbHelper.DESCRIPTION_COLUMN, "An alpaca looks like a llama.");
newValues.put(ZooDbHelper.FILE_PATH_COLUMN, "/storage/alpaca.png");
db.insert(ZooDbHelper.DATABASE_TABLE, null, newValues);
```

File location

Internal storage (by default) path:
/data/data/$PACKAGE_NAME/databases

SQL Queries

SQL query to select field with a specific field value:

```
SELECT field1, field2
FROM table
WHERE field1 = value
```

Example:

```
SELECT name, description
FROM Zoo
WHERE name = 'alpaca';
```

Querying

```
String where = null;
String whereArgs[] = null;
String groupBy = null;
String having = null;
String order = null;

String[] resultColumns = {ZooDbHelper.ID_COLUMN, ZooDbHelper.NAME_COLUMN,
ZooDbHelper.DESCRIPTION_COLUMN, ZooDbHelper.FILE_PATH_COLUMN};

Cursor cursor = db.query(ZooDbHelper.DATABASE_TABLE, resultColumns, where, whereArgs,
groupBy, having, order);

while (cursor.moveToNext()) {
          int id = cursor.getInt(0);
          String name = cursor.getString(1);
          String description = cursor.getString(2);
          String filepath = cursor.getString(3);
          Log.d("ZOO", String.format("%s,%s,%s,%s", id, name, description, filepath));
}
```

Deleting

```
// Define 'where' part of query.
String whereClause = ZooDbHelper.ID_COLUMN + "=?";

// Specify arguments in placeholder order.
String[] whereArgs = {"4"};

// Issue SQL statement
db.delete(ZooDbHelper.DATABASE_TABLE, whereClause, whereArgs);
```

Updating

```
String whereClause = ZooDbHelper.ID_COLUMN + "= ?";
String[] whereArgs = {"1"};

ContentValues newValues = new ContentValues();
newValues.put(ZooDbHelper.NAME_COLUMN, "alpaca");
newValues.put(ZooDbHelper.DESCRIPTION_COLUMN, "An alpaca is cute.");
newValues.put(ZooDbHelper.FILE_PATH_COLUMN, "/storage/alpaca.png");

db.update(ZooDbHelper.DATABASE_TABLE, newValues, whereClause, whereArgs);
```

Persisting Database Connection

Since getWritableDatabase() and getReadableDatabase() are expensive to call when the database is closed, you should leave your database connection open for as long as you possibly need to access it.
Typically, it is optimal to close the database in the onDestroy() of the calling Activity.

```
@Override
protected void onDestroy() {
    mDbHelper.close();
    super.onDestroy();
}
```

https://developer.android.com/training/basics/data-storage/databases.html

Network Storage

Network Storage

Save and retrieve data over the network

Advantage

- Everything backed up in cloud
- Easy syncing between devices

Disadvantages

- Requires network connection
- Adds latency

Mobile Backend-as-a-Service (MBaaS)

Cloud services designed for mobile:

- storage
- user management
- push notifications
- social networking services
- analytics

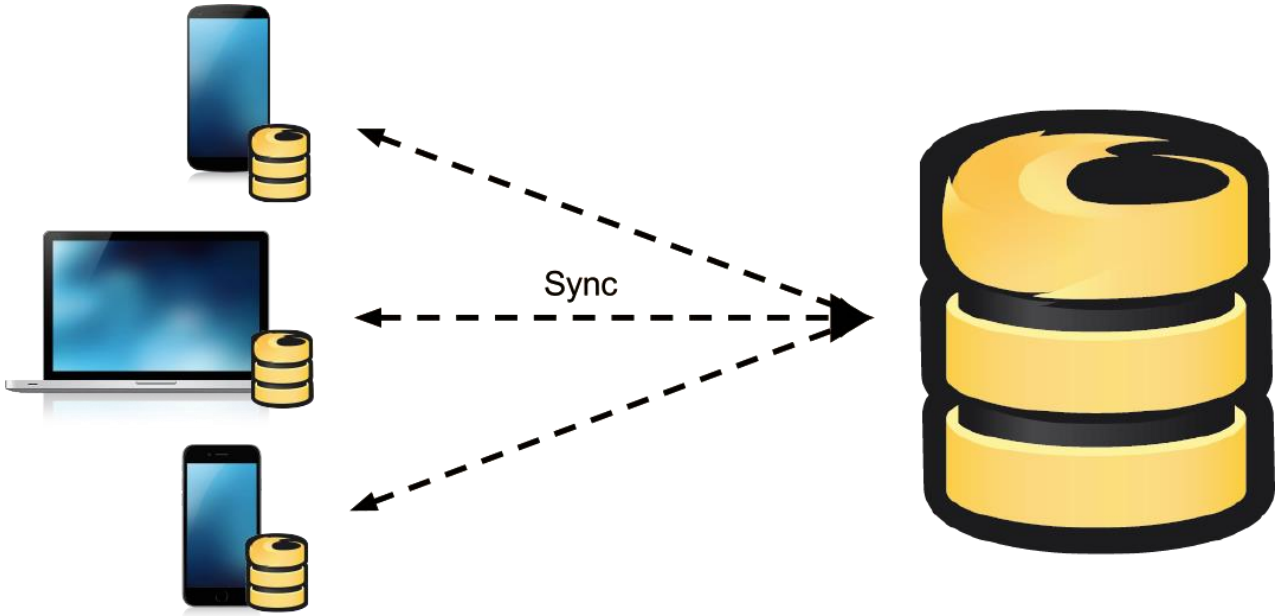
Cross-platform APIs: Android, iOS, web

MBaaS comparison spreadsheet

Firebase

- Realtime database which provides an API that allows developers to store and sync data across multiple clients
- Acquired by Google in 2014

Firebase



Firebase

Demo: Real-time updates

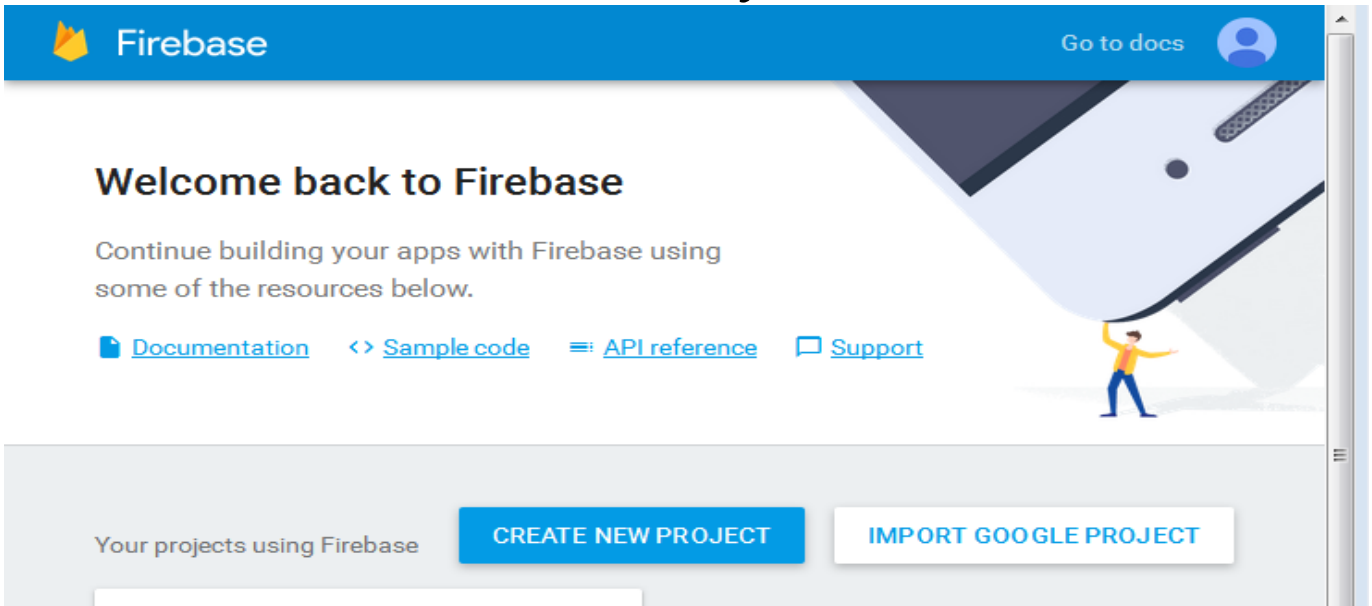
[Firebase Android Quick Start Guide]

Web view of data: [https://intense-torch-2798.firebaseio.com/]

Firebase Project Set up

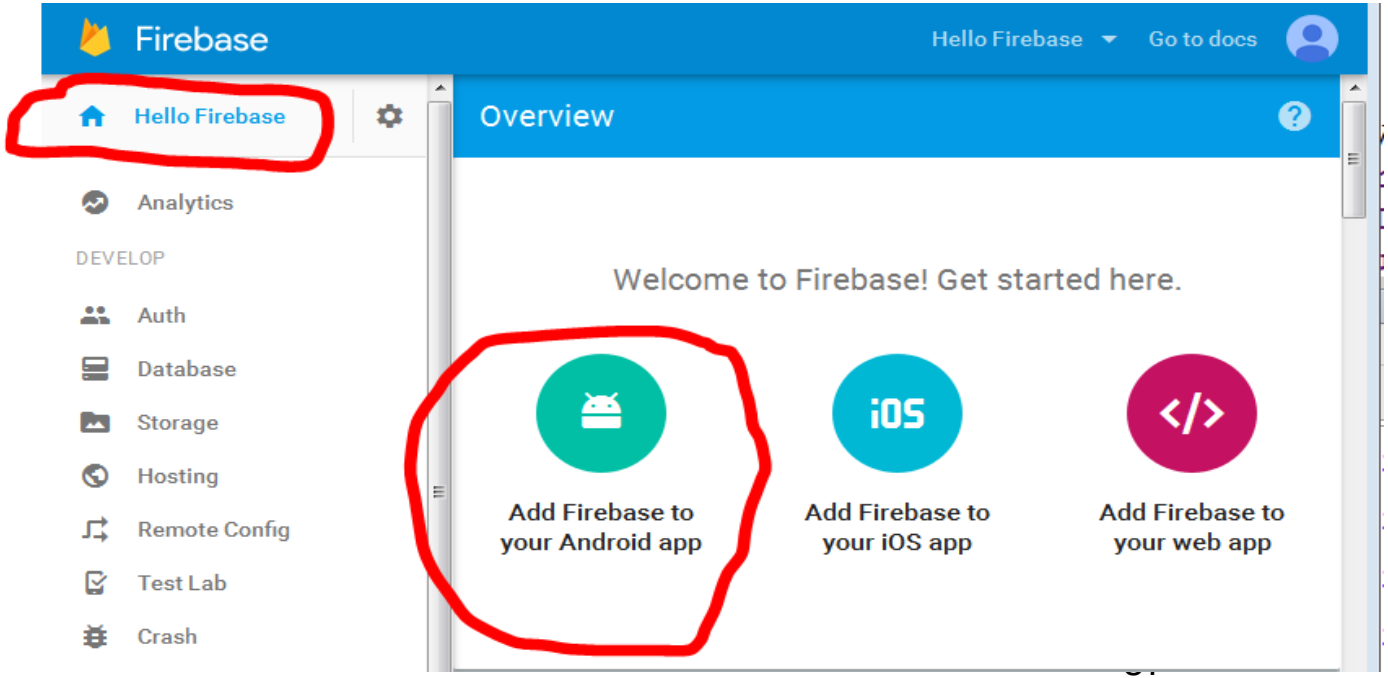
Create Firebase project in console

Just needs name and country



Firebase Project Console

After creating project, overview page:



Firebase for Android Project

Adding Firebase to Android app

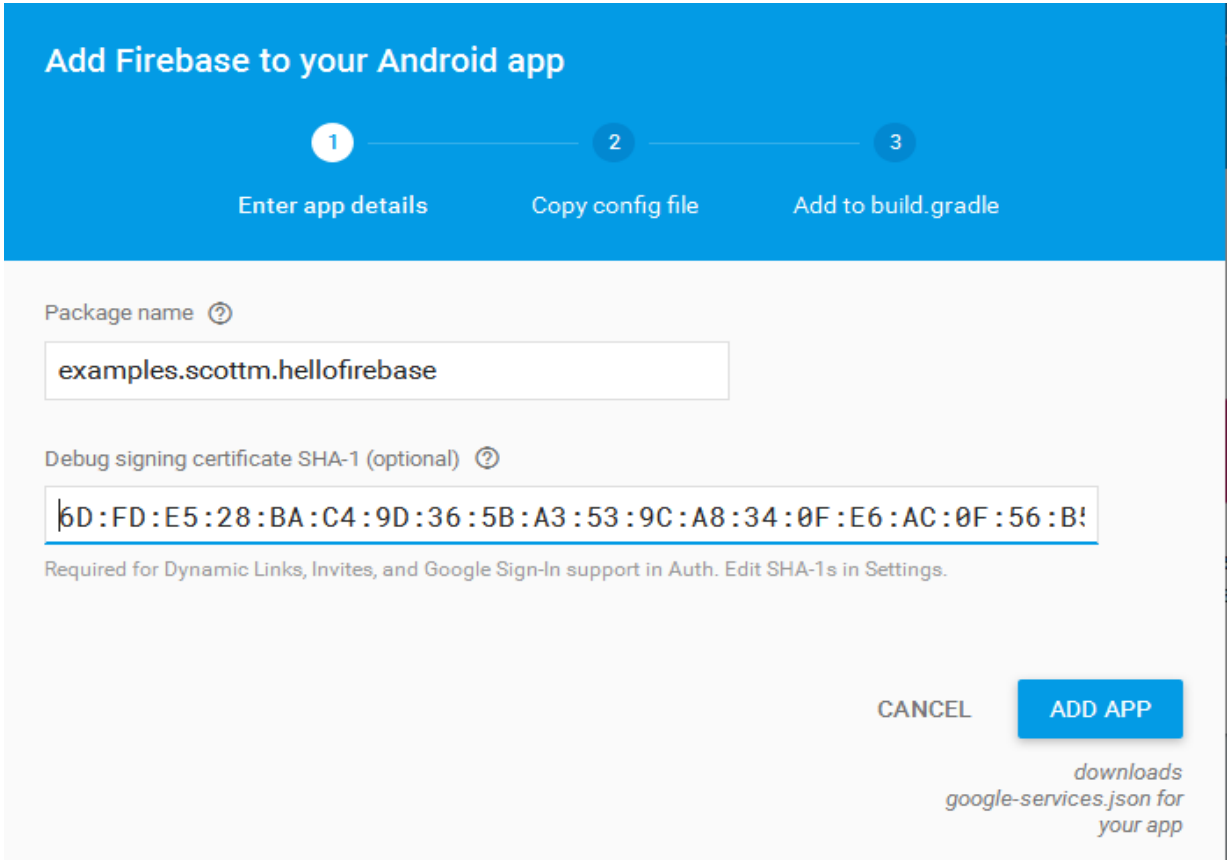
Need package name (easy)

Debug signing certificate SHA-1 hash (for use of some Firebase features)

Uses the keytool program included with Java

"Manages a keystore (database) of cryptographic keys, X.509 certificate chains, and trusted certificates. "

Adding Firebase to Android App



Firebase Config File for App

After providing package name and SHA-1 fingerprint

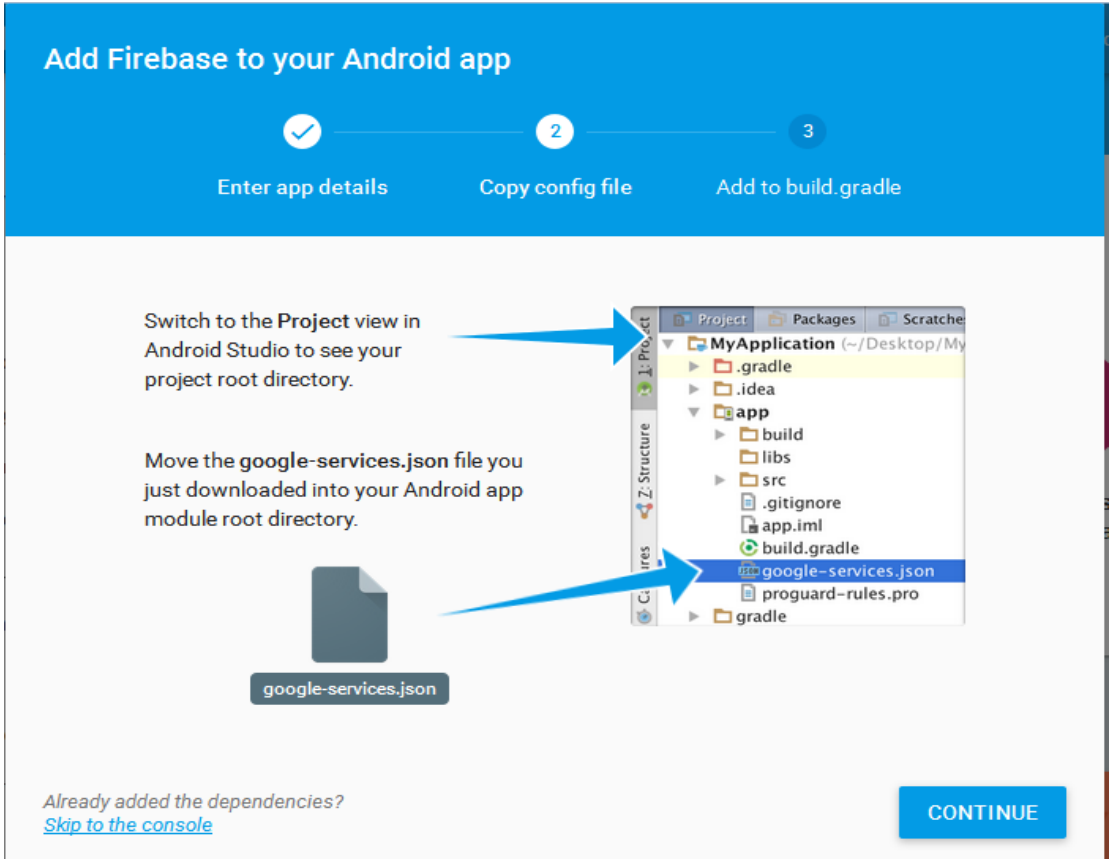
...

Firebase generates a JSON file named google-services.json specific for this project

multiple projects / apps -> repeat steps

Download and add file to project

Firebase Config File for App



google-services.json

```
{
  "project_info": {
    "project_number": "489833291042",
    "firebase_url": "https://hello-firebase-cb60f.firebaseio.co
    "project_id": "hello-firebase-cb60f",
    "storage_bucket": "hello-firebase-cb60f.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:489833291042:android:69b93ad9212
        "android_client_info": {
          "package_name": "examples.scottm.hellofirebase"
        }
      },
      "oauth_client": [
        {
          "client_id": "489833291042-ecutirgvod48scbcs6obrllsaq
          "client_type": 1,
          "android_info": {
```

Update Gradle Files

The Google services plugin for Gradle loads the `google-services.json` file you just downloaded. Modify your build.gradle files to use the plugin.

1. **Project-level build.gradle** (`<project>/build.gradle`):

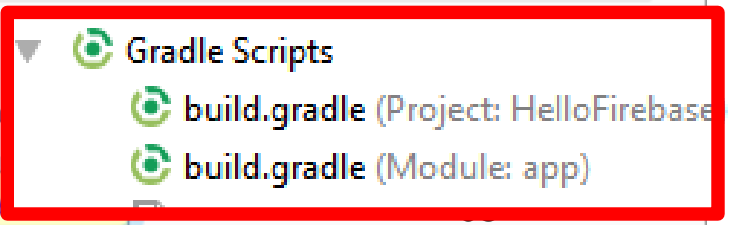
```
buildscript {
  dependencies {
    // Add this line
    classpath 'com.google.gms:google-services:3.0.0'
  }
}
```

2. **App-level build.gradle** (`<project>/<app-module>/build.gradle`):

```
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

includes Firebase Analytics by default

3. Finally, press "Sync now" in the bar that appea



Gradle files have changed sir

Firebase Data Model

All Firebase database data is stored as JSON objects.

There are no tables or records.

Firebase Data Model

```
// Example Storage
{
  "50_percent_more": {
    "name": "Kanye",
    "age": 38
  },
  "pharma_bro": {
    "name": "Martin Shkreli",
    "age": 32
  }
}
```

In Java, the JSON tree is translated into one of several types of objects:

`String`

`Boolean`

`Long`

`Double`

Firebase - Data Representation

You can define a class which Firebase will automatically map to JSON

```
public class User {
   private int age;
   private String name;

   public User() {}

   public User(String name, int age) {
       this.name = name;
       this.age = age;
   }

   public long getAge() {
       return age;
   }

   public String getName() {
       return name;
   }
}
```

Firebase - Saving Object Data

```
Firebase alanRef = ref.child("users").child("50_percent_more");
User alan = new User("Kanye", 38);
alanRef.setValue(alan);

// Result
{
  "users": {
    "50_percent_more": {
      "age": "38",
      "name": "Kanye"
    }
  }
}
```

Firebase Capabilities

Firebase has a host of capabilities
User authorization
database storage
storage for larger files
cloud messaging
push notifications
analytics
hosting of web content

Summary

Use Shared Preferences for primitive data
Use internal device storage for private data
Use external storage for large data sets that are not private
Use SQLite databases for structured storage
Firebase