CREATE-BEST

Module 1 – Creating a Cloud database for your Android Application

By: Miguel Garzón
http://www.site.uottawa.ca/~mgarz042/

uOttawa

Agenda

- Database Introduction
- Motivation
- Development of a Back-end
  - Cloud Server
  - Custom Server
  - Mobile Backend as a Service
- Overview of Firebase
- Basics of JSON objects
- Creating a Firebase database
- Reading and Saving data
- Basic Firebase Example

Database Introduction

- A database is a collection of information that is organized in a way to make it easy to access, manipulate, and update this information.

- Databases are an integral part of most software systems (web, mobile..)

- Two main types of databases:

  - Relational database systems (SQL)

  - NOSQL

Database Introduction – Relational DBMS

- Information is stored in tables

- There are **relations** between tables

- Two main concepts

  - **Primary Key** (PK): A column in a table that gives a unique identifier for every row in that given table

  - **Foreign Key**: A column in a table that references a row in another table via the referenced row's primary key

Relational Database Example
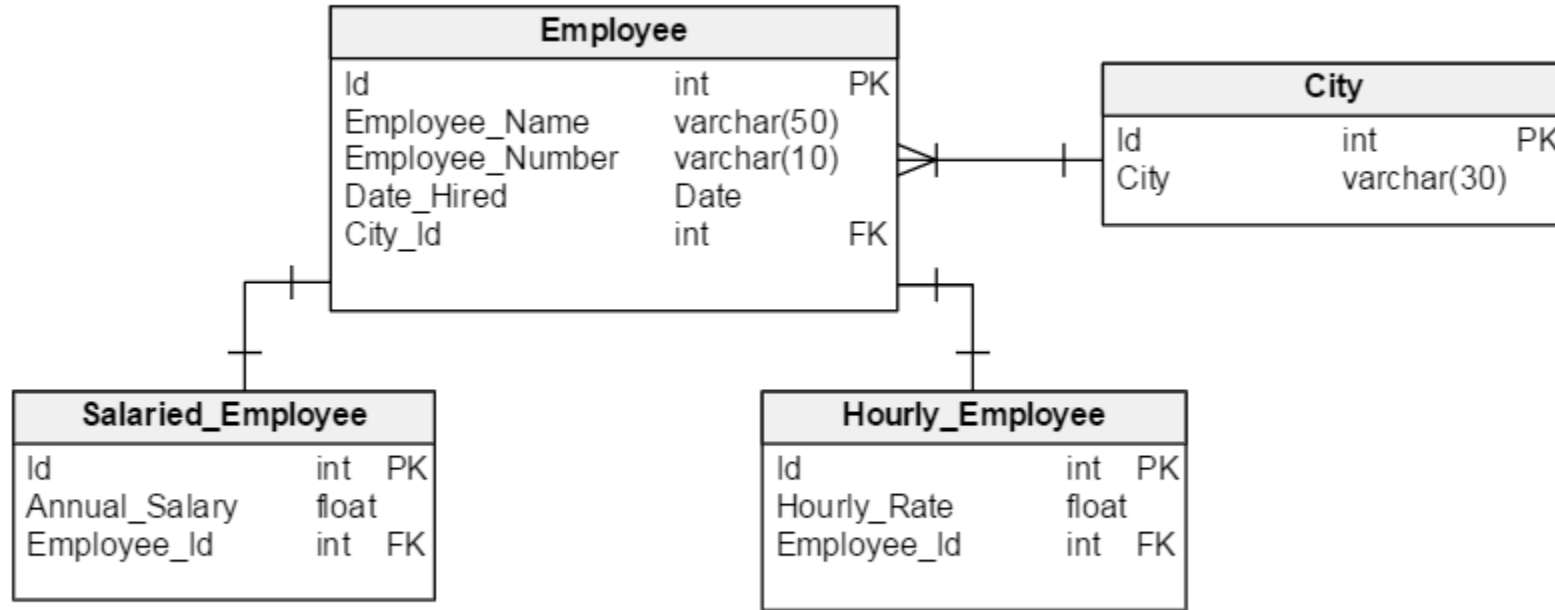
NOSQL Database

- Relatively new, robust approach to database design

- Addresses one of the main issues plaguing RDBMS ➔ SCALABILITY!

- Ideal for storing unstructured data, and data with no relations

  – User and session data

  – Chat

  – Messaging

  – Log data

Motivation

- If you are planning to develop a mobile app, chances are that you will require a **backend** for **storing** information, or **managing** content.

- Depending on your needs, your knowledge and the project requirements, you have several options for developing a backend:
  1. **Custom server**: You can set up your own server and host the backend on it. You will have complete control over the server and you can customize it to your specific requirements.
  2. **Cloud servers:** Cloud servers mean virtual servers which run on cloud computing environment. Amazon AWS, Google App Engine and Azure are the most popular cloud server options.
  3. **Mobile Backend as a Service (MBaaS):** BaaS enables you to manage a centralised database that lets your users share content via the cloud.
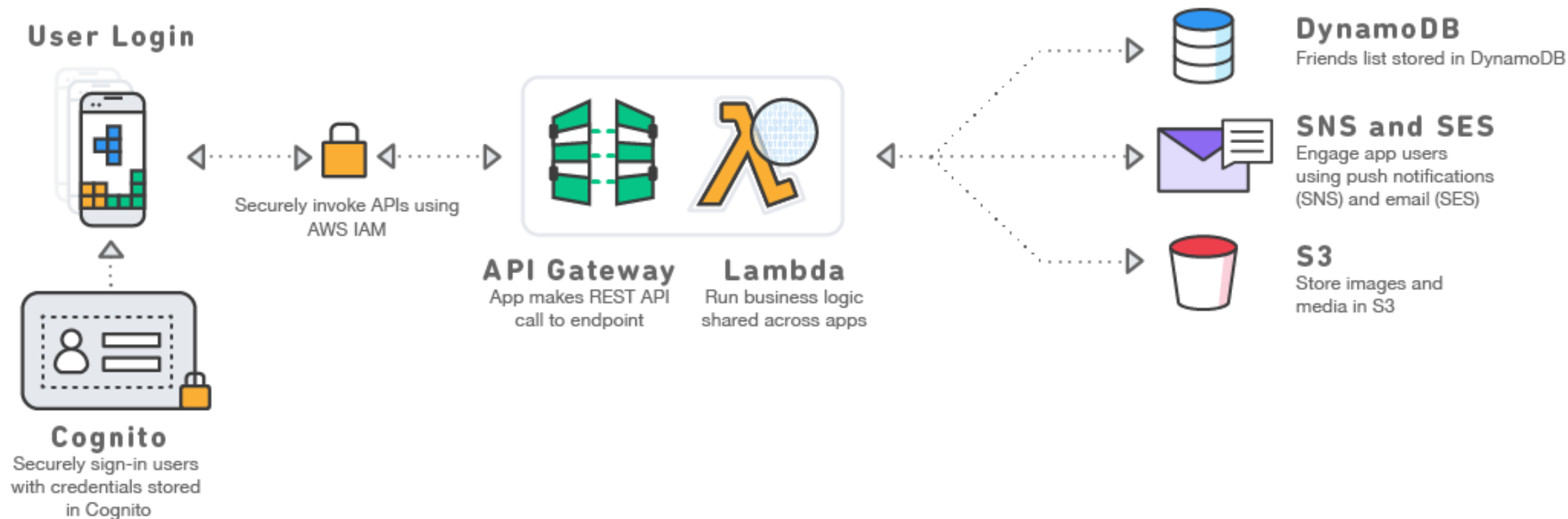
Custom Server

- **What is required?**
  1. Select the technology to be used for the server-side environment.
  2. Setting up the environment
  3. Develop an API (REST?)
  4. Add Authentication
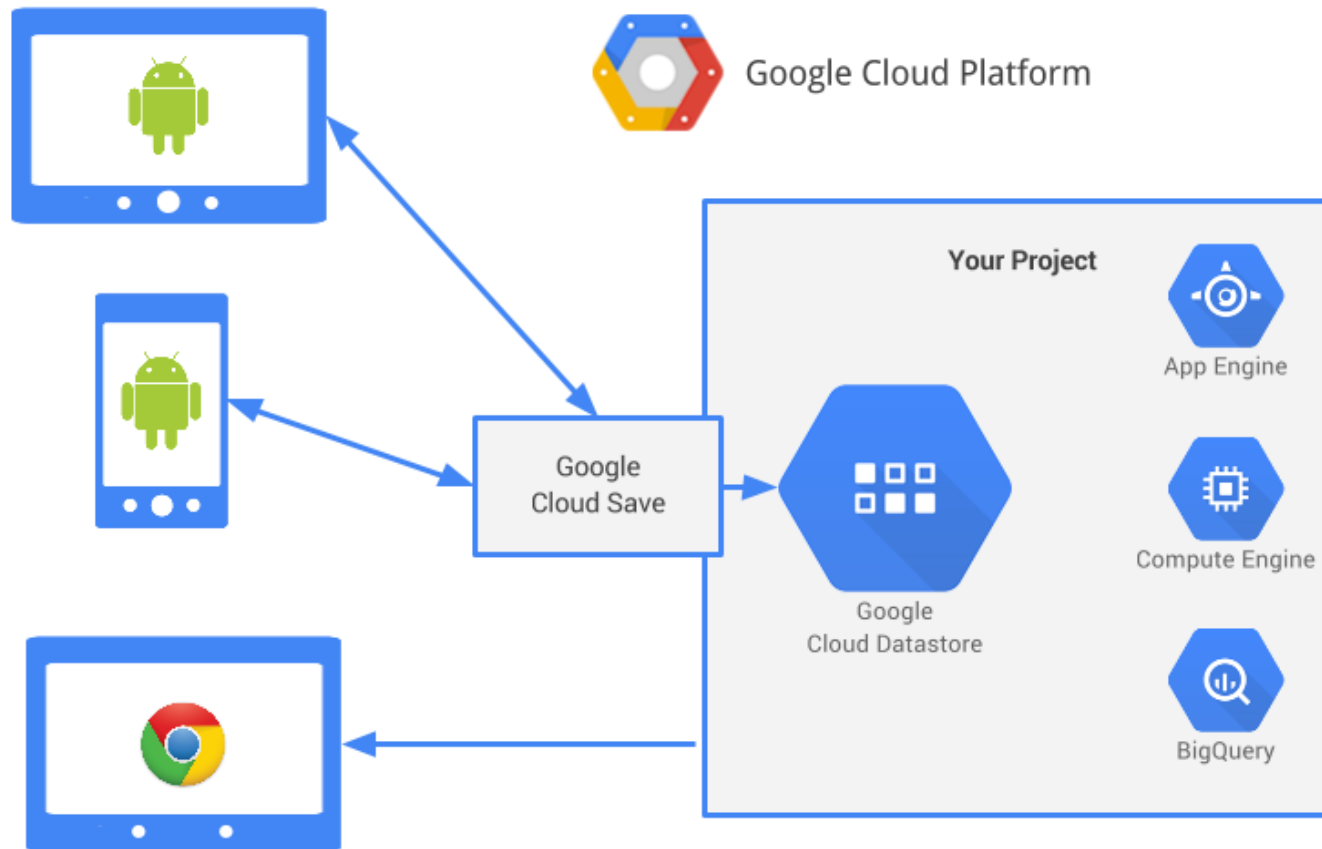 5. Handle security, routes, etc.
  6. And more..

Cloud Server – AWS

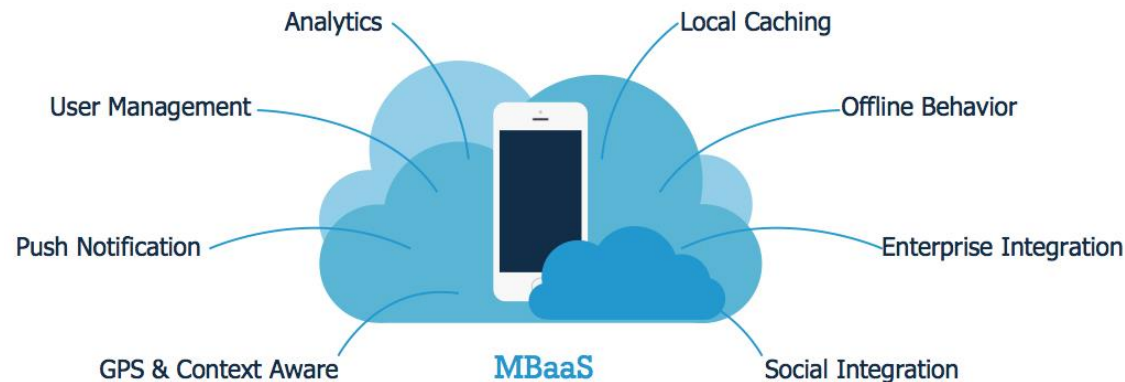# Cloud Server – Google Cloud Platform

Mobile Backend as a Service (MBaaS)

- The basic idea is similar to having your back-end development, maintenance and management outsourced to another party.

In other words,

The back-end is made available to developers as a web service.

Ever Heard of Firebase?

- **Firebase**, a popular backend platform acquired by Google in October 2014.
- Firebase allows you to store and sync data to a **NoSQL cloud database**.
  - The data is stored as **JSON**, synced to all connected clients in realtime, and available when your app goes offline.
  - It offers APIs that enable you to **authenticate** users with email and password, Facebook, Twitter, GitHub, Google, anonymous auth, or to integrate with existing authentication system.
  - Other than the Realtime Database and Authentication, it offers a myriad of other services including Cloud **Messaging**, **Storage**, **Hosting**, **Remote Config**, Test Lab, Crash Reporting, Notification, App Indexing, Dynamic Links, Invites, AdWords, AdMob.

Firebase Data Basics – JSON objects
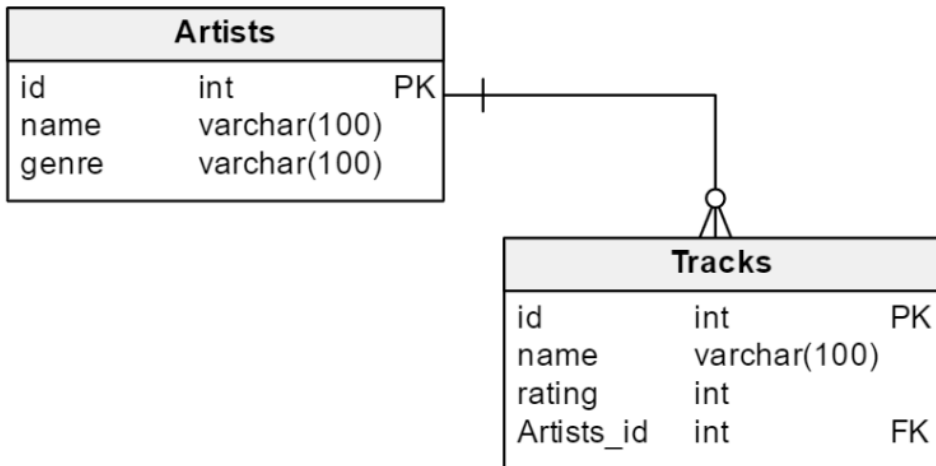
NON SQL

```
{
    "Artists":{
        "artist_id_1":{
            "name":"Atif Aslam",
            "genre":"Rock"
        },
        "artist_id_2":{
            "name":"Arijit Singh",
            "genre":"Rock"
        }
    },

    "Tracks":{
        "artist_id_1":{
            "track_id_1":{
                "name":"Taj dar e haram",
                "rating":5
            }
        }
    }
}
```

Artists

id	int	PK
name	varchar(100)	
genre	varchar(100)	

Tracks

id	int	PK
name	varchar(100)	
rating	int	
Artists_id	int	FK

SQL

Firebase Terminology

- All Firebase Realtime Database data is stored as JSON objects.

```
productcatalog-d825a
  products
    -Kk9FXCvnq2WvdZ4m_jA
        id: "-Kk9FXCvnq2WvdZ4m_jA
        price: 250
        productName: "Monitor Dell"
    -Kk9FZDQIcV_eiYg74NS
        id: "-Kk9FZDQIcV_eiYg74NS
        price: 25
        productName: "Mouse"
    -Kk9Fhhh_fHqWzxmyOZz
        id: "-Kk9Fhhh_fHqWzxmyOZz
        price: 1800
        productName: "Dell XPS I34"
```
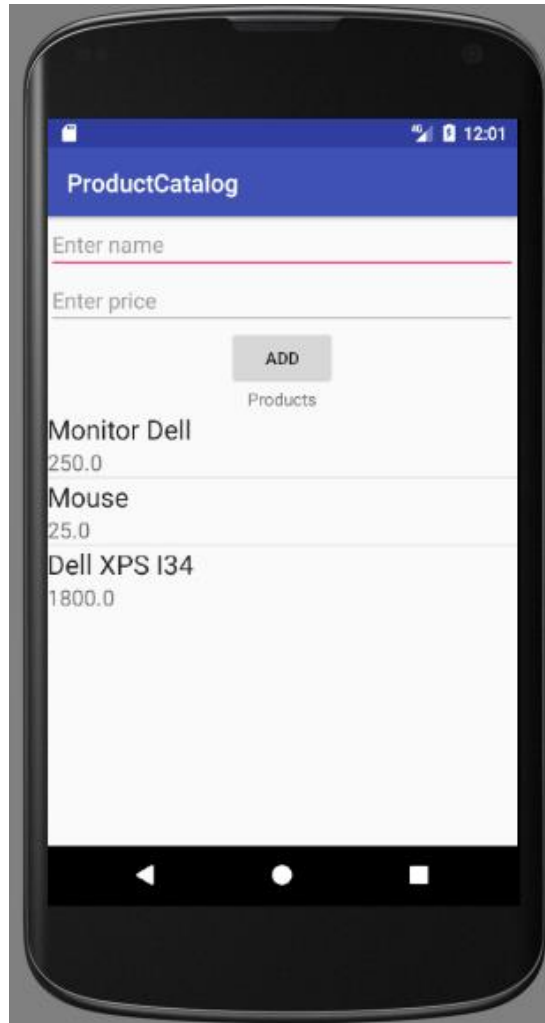
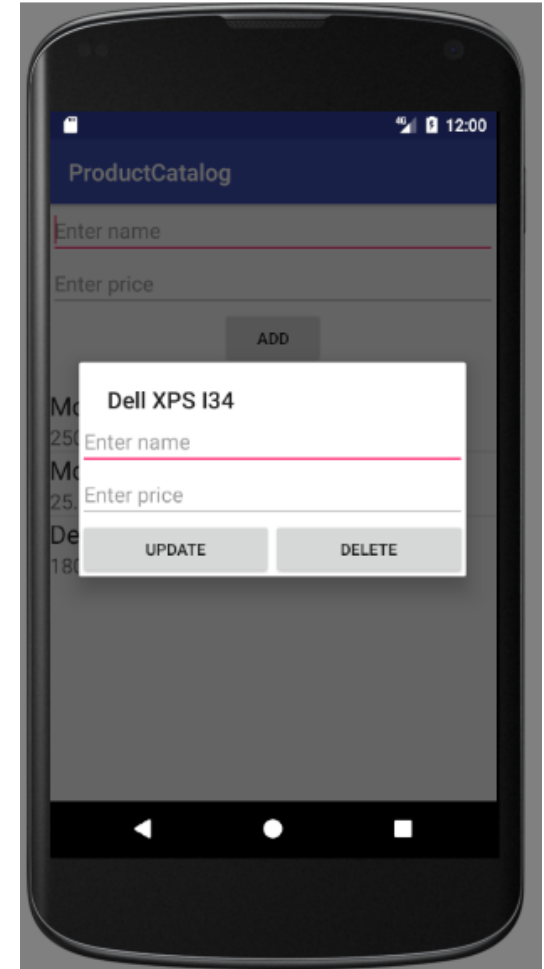# BASIC FIREBASE EXAMPLE

User Interface

Main Interface:
RelativeLayout
-> EditText
-> EditText
-> Button
-> TextView
-> ListView

Update Dialog:
LinearLayout
-> EditText
-> EditText
-> LinearLayout
-> Button
-> Button

STEP 1 – Make an Account

- The **free account** gives you the ability to have 100 devices connect to Firebase at a single point in time.

Products	Spark Plan Generous limits for hobbyists Free	Flame Plan Predictable pricing for growing apps $25/month	Blaze Plan Calculate pricing for apps at scale Pay as you go
Analytics, App Indexing, Dynamic Links, Invites, Remote Config, Cloud Messaging, Authentication, and Crash Reporting.	✓ Included	✓ Included	✓ Included
Realtime Database			
Simultaneous connections	100	Unlimited	Unlimited
GB stored	1GB	2.5GB	$5/GB
GB downloaded	10GB/month	20GB/month	$1/GB
Automated backups	✗	✗	✓

It's worth noting that 99% of apps never outgrow the free tier, so it's a great tier to start in.

17

STEP 1 – Make an Account (cont'd)

- The **free account** gives you the ability to have 100 devices connect to Firebase at a single point in time. It's worth noting that 99% of apps never outgrow the free tier, so it's a great tier to start in.

1. Go to **firebase.google.com** and **create an account**.
2. After **logging** in to your account, head over to the **Firebase console**
3. And **create a project** that will hold your app's data.

STEP 2: Connect Firebase to Your Application

- To add Firebase to your android application:

  2a. You can follow a **manual** process (three main steps are required).

  Or

  2b. You can explore and integrate Firebase services in your app directly from **Android Studio** using the **Assistant** window.

If you're using the latest version of Android Studio (version 2.2 or later), we recommend using the Firebase Assistant to connect your app to Firebase.

- **Prerequisites**
  - A device running Android 4.0 (Ice Cream Sandwich) or newer, and Google Play services 10.2.4 or higher
  - The Google Play services SDK from the **Google Repository**, available in the Android SDK Manager
  - The latest version of Android Studio, version 1.5 or higher. **Most recent version is 2.3.1.**

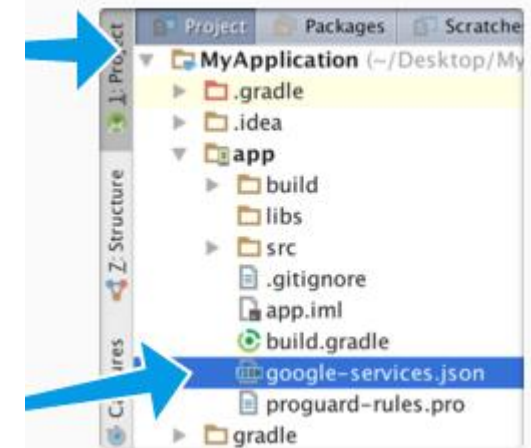# STEP 2a: <u>Manually</u> Connect Firebase to Your Application
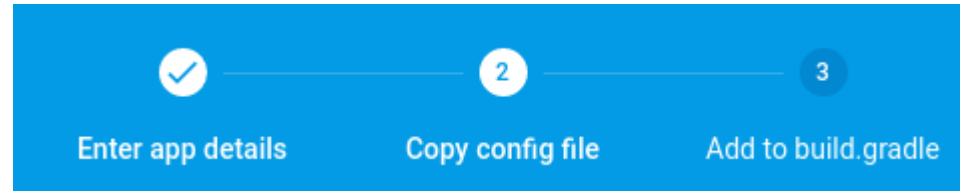
1. **Enter app details:**

    a) **Create a project:** Enter a name and country/region for the project.

    b) **Enter your Android project's package name** in the window that pops up. If your app is going to use certain Google Play services such as Google Sign-In, App Invites and Dynamic Links then you will have to provide the SHA-1 of your signing certificate. The application we will build won't be using any of these services, so leave this field empty. More info about certificates. https://developers.google.com/android/guides/client-auth

2. **Copy Config file:** The wizard will allow you to generate a configuration file. **Download** the file (google-services.json) and move it into your Android app module root directory.

The JSON file contains configuration settings that the Android app needs to communicate with the Firebase servers. It contains details such as the URL to the Firebase project, the API key.

STEP 2a: <u>Manually</u> Connect Firebase to Your Application (cont'd)

3. Add to build.gradle

The Google services plugin for Gradle 🔗 loads the `google-services.json` file that 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 appears in the IDE:

Gradle files have changed sir   Sync now

FINISH

STEP 2b: Connect Firebase to Your Application using Android Studio

First make sure you have installed Google Repository version 26 or higher, using the following steps:

- Click **Tools > Android > SDK Manager**.
- Click the **SDK Tools** tab.
- Check the **Google Repository** checkbox, and click **OK**.
- Click **OK** to install.
- Click **Background** to complete the installation in the background, or wait for the installation to complete and click **Finish**.

STEP 2b: Connect Firebase to Your Application using Android Studio (cont'd)

1. Click **Tools > Firebase** to open the **Assistant** window.

2. Click to expand one of the listed features then click the **Save and Retrieve data** (under Realtime Database) to connect to Firebase and automatically add the necessary code to your app.
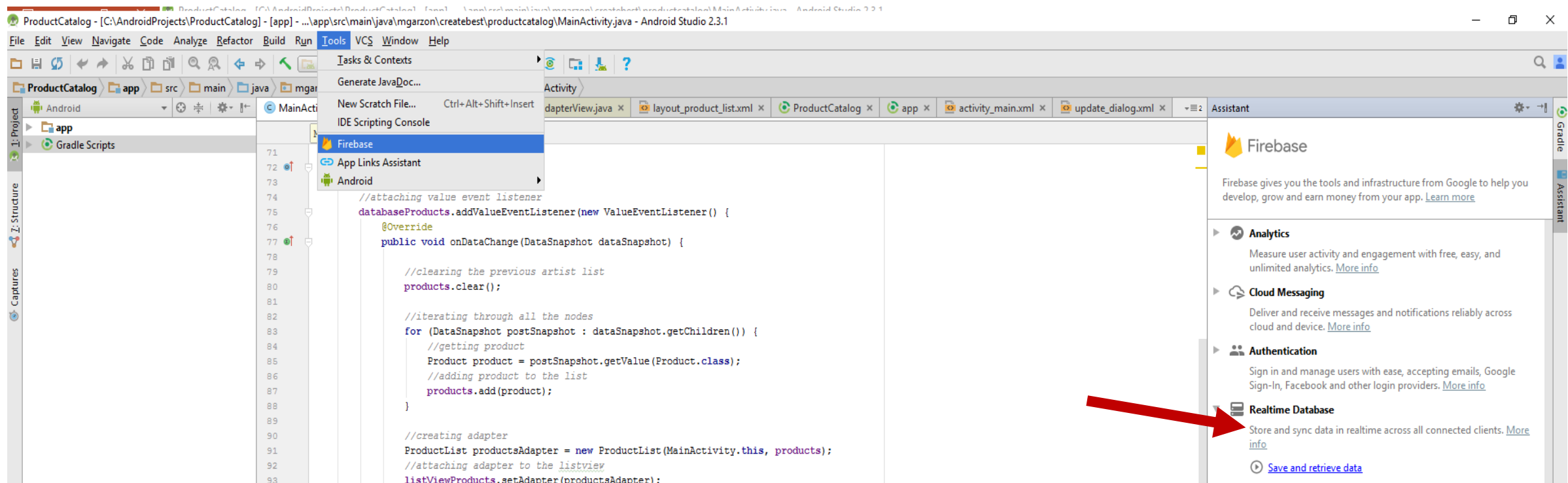
STEP 2b: Connect Firebase to Your Application using Android Studio (cont'd)

3. Click **Connect to Firebase** to open the window.
   Enter the name for your Firebase project.
4. Click **Add the Realtime Database to your app.**

You are now ready to write and read form your Database!

Save and retrieve data

Our cloud database stays synced to all connected clients in realtime and remains available when your app goes offline. Data is stored in a JSON tree structure rather than a table, eliminating the need for complex SQL queries.

Launch in browser

① Connect your app to Firebase

Connect to Firebase

② Add the Realtime Database to your app

Add the Realtime Database to your app

③ Configure Firebase Database Rules

The Realtime Database provides a declarative rules language that allows you to define how your data should be structured, how it should be indexed, and when your data can be read from and written to. By default, read and write access to your database is restricted so only authenticated users can read or write data. To get started without setting up Authentication, you can configure your rules for public access. This does make your database open to anyone, even people not using your app, so be sure to restrict your database again when you set up authentication.

④ Write to your database

Retrieve an instance of your database using getInstance() and reference the location you want to write to.

```
// Write a message to the database
FirebaseDatabase database = FirebaseDatabase.getIns
DatabaseReference myRef = database.getReference("me

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

You can save a range of data types to the database this way, including Java objects. When you save an object the responses from any getters will be saved as children of this location.

STEP 3 : Configure Firebase Database rules

- The Realtime Database provides a declarative rules language that allows you to define how your data should be structured, how it should be indexed, and when your data can be read from and written to. **By default, read and write access to your database is restricted so only authenticated users can read or write data.**

- To get started without setting up Authentication , you can configure your rules for public access.

Go to the firebase console and specify your rules as follows:

```
1 ▾   {
2 ▾       "rules": {
3             ".read": true,
4             ".write": true
5          }
6       }
```

Your turn!

Write and Read Data with Firebase

To download and set up the sample application in Android Studio:

1. **Download** the ProductCatalog sample app from my personal Github.

2. You can either use the "Download ZIP" button on the Github Page or clone on the command line: https://github.com/mgarzon/ProductCatalog_v1

   git clone https://github.com/mgarzon/ProductCatalog_v1.git

3. **Import** the project in Android Studio: Click File > New > Import Project.

4. **Implement** the code to **add**, **delete** and **update** a *product* which will be stored at the Firebase database.

Import Required Libraries

- Import libraries to use Text Boxes, Spinners, Buttons and required Database libraries such as FirebaseDatabase.

```
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.util.ArrayList;
import java.util.List;
```

Steps to Read and Write Data on Android

- Get a DatabaseReference
  - To read or write data from the database, you need an instance of DatabaseReference:

```
DatabaseReference databaseProducts;
```

  - Add the following line to the OnCreate method.

```
databaseProducts = FirebaseDatabase.getInstance().getReference("products");
```

Find the full documentation about reading and writing data on Android here:

Listen for Value Events

- To read data at a path and listen for changes, use the addValueEventListener() or addListenerForSingleValueEvent() method to add a ValueEventListener to a DatabaseReference.

```java
protected void onStart() {
    super.onStart();
    //attaching value event listener
    databaseProducts.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {

            //..
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {

        }
    });
}
```

Listen for Value Events (cont'd).

- Add the following to the onDataChange;
  - Clear the previous artis list.

```
//clearing the previous artist list
products.clear();
```

  - Iterate through with the following.

```
//iterating through all the nodes
for (DataSnapshot postSnapshot : dataSnapshot.getChildren()) {
    //getting product
    Product product = postSnapshot.getValue(Product.class);
    //adding product to the list
    products.add(product);
}
```

  - Finally, create the adapter.

```
//creating adapter
ProductList productsAdapter = new ProductList(MainActivity.this, products);
//attaching adapter to the listview
listViewProducts.setAdapter(productsAdapter);
```

Listen for Value Events (final look)

```java
//attaching value event listener
databaseProducts.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {

        //clearing the previous artist list
        products.clear();

        //iterating through all the nodes
        for (DataSnapshot postSnapshot : dataSnapshot.getChildren()) {
            //getting product
            Product product = postSnapshot.getValue(Product.class);
            //adding product to the list
            products.add(product);
        }
        //creating adapter
        ProductList productsAdapter = new ProductList(MainActivity.this, products);
        //attaching adapter to the listview
        listViewProducts.setAdapter(productsAdapter);
    }
    @Override
    public void onCancelled(DatabaseError databaseError) {

    }
});
```

CREATEBEST - Firebase

Read and Write Data

- Basic write operations
- For basic write operations, you can use setValue() to save data to a specified reference, replacing any existing data at that path. You can use this method to:
- Pass types that correspond to the available JSON types as follows:
  - String
  - Long
  - Double
  - Boolean
  - Map<String, Object>
  - List<Object>

Add a Product to the Database

- Get the values from the TextBoxes for name and the price.

```java
private void addProduct() {
    //getting the values to save
    String name = editTextName.getText().toString().trim();
    double price = Double.parseDouble(String.valueOf(editTextPrice.getText().toString()));

    //..
}
```

- Inside the addProduct, check the boxes if the values are provided. If the textboxes contain values, do the following and print a success message.

```java
    //checking if the value is provided
    if (!TextUtils.isEmpty(name)) {

        //..

        //displaying a success toast
        Toast.makeText(this, "Product added", Toast.LENGTH_LONG).show();
    } else {
        //if the value is not given displaying a toast
        Toast.makeText(this, "Please enter a name", Toast.LENGTH_LONG).show();
    }
```

Add a Product to the Database (cont'd).

- Get a unique id for the each product to be saved to the database.

```java
//getting a unique id using push().getKey() method
//it will create a unique id and we will use it as the Primary Key for our Product
String id = databaseProducts.push().getKey();
```

- Create a Product object and save this object.

```java
//creating a Product Object
Product product = new Product(id, name, price);

//Saving the Product
databaseProducts.child(id).setValue(product);
```

- Clear the TextBoxes.

```java
//setting edittext to blank again
editTextName.setText("");
editTextPrice.setText("");
```

Add a Product to the Database (final look)

```java
private void addProduct() {
    //getting the values to save
    String name = editTextName.getText().toString().trim();
    double price = Double.parseDouble(String.valueOf(editTextPrice.getText().toString()));

    //checking if the value is provided
    if (!TextUtils.isEmpty(name)) {

        //getting a unique id using push().getKey() method
        //it will create a unique id and we will use it as the Primary Key for our Product
        String id = databaseProducts.push().getKey();

        //creating a Product Object
        Product product = new Product(id, name, price);

        //Saving the Product
        databaseProducts.child(id).setValue(product);

        //setting edittext to blank again
        editTextName.setText("");
        editTextPrice.setText("");

        //displaying a success toast
        Toast.makeText(this, "Product added", Toast.LENGTH_LONG).show();
    } else {
        //if the value is not given displaying a toast
        Toast.makeText(this, "Please enter a name", Toast.LENGTH_LONG).show();
    }
}
```

36

Update a Product on the Database

- Get the string id, name and the price.

```java
private void updateProduct(String id, String name, double price) {
    //..
}
```

- As the unique id, get the product reference.

```java
//getting the specified product reference
DatabaseReference dR = FirebaseDatabase.getInstance().getReference("products").child(id);
```

- Update the product by using setValue().

```java
//updating product
Product product = new Product(id, name, price);
dR.setValue(product);
```

- Show a success message.

```java
Toast.makeText(getApplicationContext(), "Product Updated", Toast.LENGTH_LONG).show();
```

Update a Product on the Database (final look)

```java
private void updateProduct(String id, String name, double price) {
    //getting the specified product reference
    DatabaseReference dR = FirebaseDatabase.getInstance().getReference("products").child(id);
    //updating product
    Product product = new Product(id, name, price);
    dR.setValue(product);

    Toast.makeText(getApplicationContext(), "Product Updated", Toast.LENGTH_LONG).show();
}
```

Delete a Product from the Database

- As the unique id, get the product reference.

```
private boolean deleteProduct(String id) {
    //getting the specified product reference
    DatabaseReference dR = FirebaseDatabase.getInstance().getReference("products").child(id);
```

- Remove the product and show a success message.

```
    //removing prodct
    dR.removeValue();
    Toast.makeText(getApplicationContext(), "Product Deleted", Toast.LENGTH_LONG).show();
    return true;
}
```

Delete a Product from the Database (final look)

```java
private boolean deleteProduct(String id) {
    //getting the specified product reference
    DatabaseReference dR = FirebaseDatabase.getInstance().getReference("products").child(id);
    //removing prodct
    dR.removeValue();
    Toast.makeText(getApplicationContext(), "Product Deleted", Toast.LENGTH_LONG).show();
    return true;
}
```

THANK YOU!