



FREE eBook

LEARNING firebase

Free unaffiliated eBook created from
Stack Overflow contributors.

#firebase

Table of Contents

About.....	1
Chapter 1: Getting started with firebase.....	2
Remarks.....	2
Versions.....	2
Examples.....	2
Add Firebase to Your Android Project.....	2
Add Firebase to your app.....	2
Add the SDK.....	3
Setting up Firebase for IOS.....	4
Getting started in Firebase with a simple Hello World web app in JavaScript.....	12
Lets get started.....	12
Chapter 2: Cloud Functions for Firebase.....	18
Introduction.....	18
Examples.....	18
Send welcome notification emails to the users for subscribing.....	18
Now go to your Firebase Console.....	18
Install Firebase CLI in your computer.....	18
Set Google Cloud environment variables.....	19
Deploy the project and test.....	19
Chapter 3: Crash Reporting.....	21
Remarks.....	21
Official Documetantion.....	21
Examples.....	21
Setup Crash Reporting in Android.....	21
Report the error in Android.....	21
Chapter 4: Database Rules.....	23
Introduction.....	23
Remarks.....	23
Official Documentation.....	23

Examples.....	23
How to configure rules.....	23
The default rules.....	23
How to set your files publicly readable and writable.....	24
How to disable read and write access.....	24
How to grant access only to authenticated users.....	24
How to allow reading specific item from group, but prevent listing group members.....	25
Chapter 5: Email Verification after Sign Up.....	26
Syntax.....	26
Parameters.....	26
Remarks.....	26
Examples.....	26
Send-cum-Process Verification Action Code - AngularJS.....	26
Chapter 6: Firebase Realtime Database with Android.....	28
Examples.....	28
How to connect Realtime database with Android Application.....	28
Chapter 7: Firebase Console.....	30
Syntax.....	30
Parameters.....	30
Remarks.....	30
Examples.....	30
Firebase All In One.....	30
Chapter 8: Firebase Offline Capabilities.....	31
Introduction.....	31
Remarks.....	31
Examples.....	32
Enable disk persistence (Android / iOS only).....	32
Keeping data fresh (Android/iOs Only).....	32
Chapter 9: Firebase Queue.....	34
Examples.....	34
How to use firebase queue as a backend for your application.....	34

Prerequisites.....	34
Architecture.....	34
Chapter 10: FirebaseUI.....	38
Remarks.....	38
Examples.....	38
Getting Started with FirebaseUI.....	38
Chapter 11: FirebaseUI (Android).....	40
Examples.....	40
Adding the dependencies.....	40
Populating a ListView.....	40
Chapter 12: How do I listen for errors when accessing the database?.....	42
Introduction.....	42
Examples.....	42
Detect errors when writing a value on Android.....	42
Detect errors when reading data on Android.....	42
Detect errors when writing a value on iOS.....	43
Detecting errors when reading data in JavaScript.....	43
Detecting errors when writing a value in JavaScript.....	44
Detect errors when reading data on iOS.....	44
Chapter 13: How to get push key value from Firebase Database?.....	46
Introduction.....	46
Examples.....	46
Android Example.....	46
Chapter 14: How to use FirebaseRecyclerAdapter instead of RecyclerView?.....	47
Examples.....	47
Here is the Example for Use FirebaseAuth component FirebaseRecyclerAdapter.....	47
Chapter 15: How to use the Firebase Database to keep a list of Firebase Authentication use	53
Examples.....	53
How to save user profile data.....	53
Why save user data in the database.....	53
Handling User Account Data in the Realtime Database.....	54

Chapter 16: Push notification from custom server	55
Introduction	55
Examples	55
Firebase Cloud Messaging HTTP Protocol	55
Using Admin SDK(Node js)	56
Chapter 17: Storage	58
Remarks	58
Examples	58
Getting started on iOS	58
Prerequisites	58
Add Firebase Storage to your app	58
Set up Firebase Storage	59
Chapter 18: Structuring Data	61
Introduction	61
Examples	61
Do's and Don'ts	61
Two-Way Relationships	62
Chapter 19: Using Firebase with Node	64
Examples	64
Hello World Firebase Realtime Database in Node	64
Firebase-queue and worker	66
Credits	69

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [firebase](#)

It is an unofficial and free firebase ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official firebase.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with firebase

Remarks

[Firebase](#) is a Backend as a Service (Baas) very useful for mobile app development.

It provides many features like **Authentication & Security**, **Realtime Database & File Storage**, **Analytics**, **Push Notifications**, **AdMod** and many [others](#)

It provides the [SDK](#) for **Android**, **iOS**, **Web**, **NodeJS**, **C++** and **Java Server**

Versions

Platform SDK	Version	Release date
Firebase JavaScript SDK	3.7.0	2017-03-01
Firebase C++ SDK	3.0.0	2107-02-27
Firebase Unity SDK	3.0.0	2107-02-27
Firebase iOS SDK	3.14.0	2017-02-23
Firebase Android SDK	10.2	2017-02-15
Firebase Admin Node.js SDK	4.1.1	2017-02-14
Firebase Admin Java SDK	4.1.2	2017-02-14

Examples

Add Firebase to Your Android Project

Here the steps required to create a Firebase project and to connect with an Android app.

Add Firebase to your app

1. Create a Firebase project in the [Firebase console](#) and click **Create New Project**.
2. Click **Add Firebase to your Android app** and follow the setup steps.
3. When prompted, enter your **app's package name**.
It's important to enter the package name your app is using; this can only be set when you add an app to your Firebase project.

4. To add debug signing certificate SHA1 which is **required for Dynamic Links, Invites, and Google Sign-In support in Auth**, go to your project in Android Studio, click on `Gradle` tab on the right side of your window, click on `Refresh` button, go to `project (root) -> Tasks -> android -> signingReport`. This will generate **MD5** and **SHA1** both in `Run` tab. Copy paste SHA1 into firebase console.
5. At the end, you'll download a `google-services.json` file. You can download this file again at any time.
6. If you haven't done so already, copy this into your project's module folder, typically `app/`.

The next step is to Add the SDK to integrate the Firebase libraries in the project.

Add the SDK

To integrate the Firebase libraries into one of your own projects, you need to perform a few basic tasks to prepare your Android Studio project. You may have already done this as part of adding Firebase to your app.

1. Add rules to your root-level `build.gradle` file, to include the **google-services plugin**:

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

Then, in your module Gradle file (usually the `app/build.gradle`), add the apply plugin line at the bottom of the file to enable the Gradle plugin:

```
apply plugin: 'com.android.application'  
  
android {  
    // ...  
}  
  
dependencies {  
    // ...  
    compile 'com.google.firebase:firebase-core:9.4.0'  
}  
  
// ADD THIS AT THE BOTTOM  
apply plugin: 'com.google.gms.google-services'
```

The final step is to add the dependencies for the Firebase SDK using one or more **libraries available** for the different Firebase features.

Gradle Dependency Line	Service
com.google.firebase:firebase-core:9.4.0	Analytics
com.google.firebase:firebase-database:9.4.0	Realtime Database
com.google.firebase:firebase-storage:9.4.0	Storage
com.google.firebase:firebase-crash:9.4.0	Crash Reporting
com.google.firebase:firebase-auth:9.4.0	Authentication
com.google.firebase:firebase-messaging:9.4.0	Cloud Messaging / Notifications
com.google.firebase:firebase-config:9.4.0	Remote Config
com.google.firebase:firebase-invites:9.4.0	Invites / Dynamic Links
com.google.firebase:firebase-ads:9.4.0	AdMob
com.google.android.gms:play-services-appindexing:9.4.0	App Indexing

Setting up Firebase for IOS



1. Firstly, you want to go to firebase dashboard and create a new project using the 'Create New Project' button.

Welcome back to Firebase

Continue building your apps with Firebase using some of the resources below.

 [Documentation](#)  [Sample code](#)  [API reference](#)  [Support](#)

Your projects using Firebase CRE

BrainMatter  brainmatter-4f454.firebaseio.com iOS 1 app	fireAuth  fireauth-415f8.firebaseio.com iOS 1 app
KIKOO	loom

2. You want to create a new project by adding the name of your app for example I put mine as 'Cool app name' then choose your region and press 'Create Project'


Welcome back to Firebase

Continue building your apps with Firebase using some of the resources below.

 [Documentation](#) <> [Sample code](#)

Your projects using Firebase

BrainMatter

 brainmatter-4f454.firebaseio.com
iOS 1 app


KIKOO

loom

Create a project

Project name

Country/region 

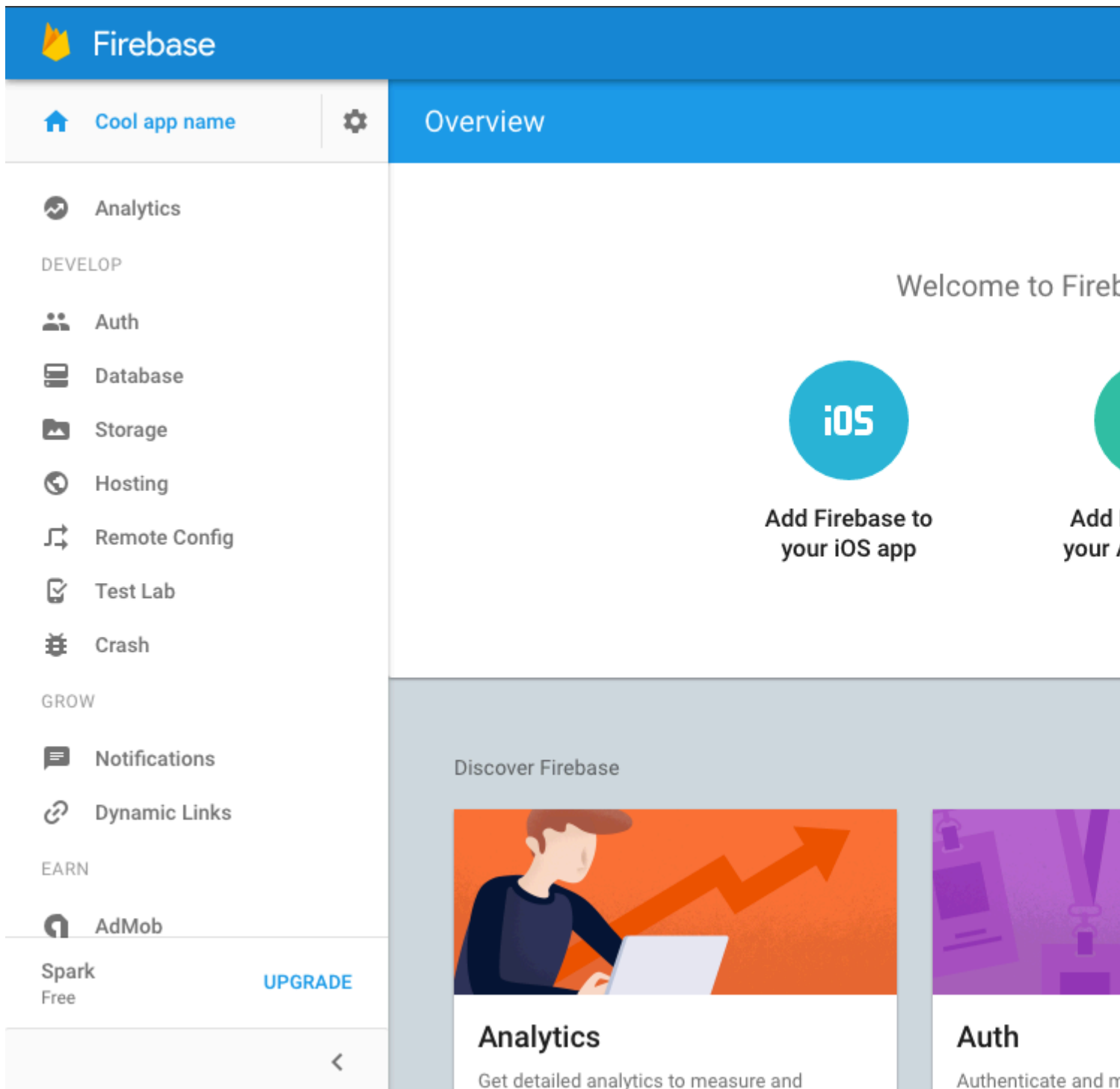
United States 

By default, your Firebase Analytics data will enhance other features and Google products. You can control how your data is shared in your settings at anytime. [Learn more](#)

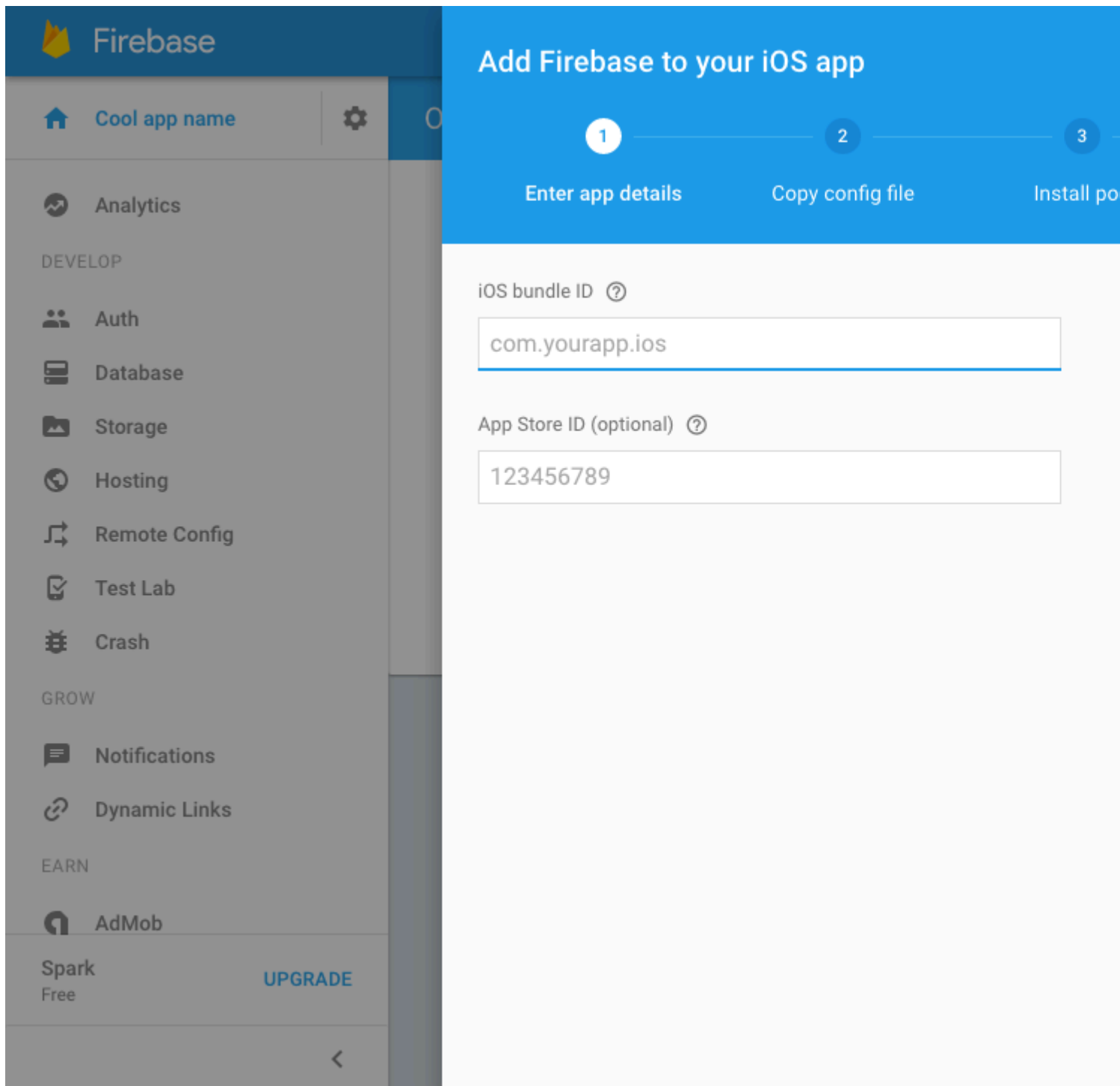
CANCEL

CREATE

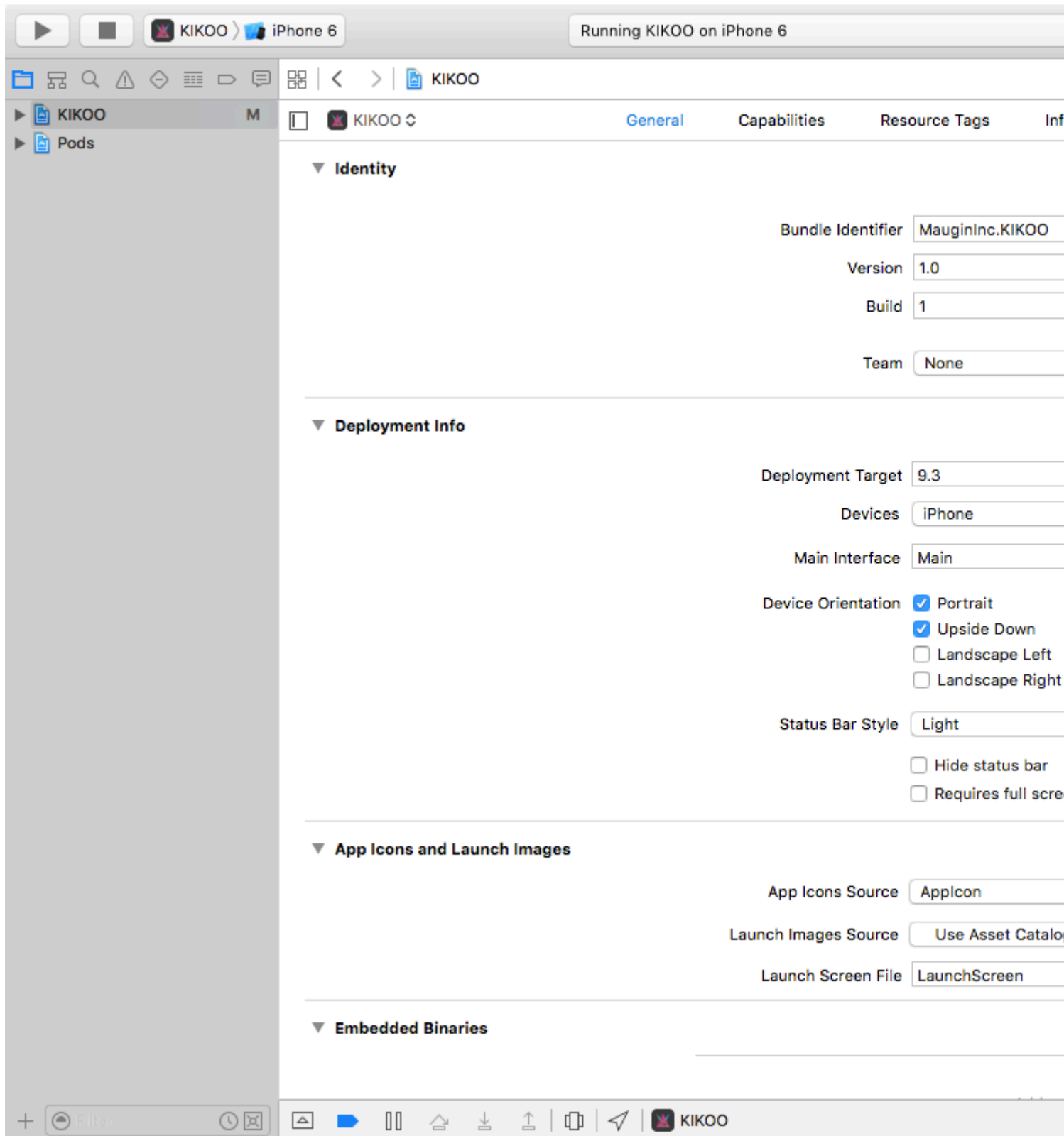
3. After creating project you will be directed to this page which is the dashboard and from here you have to pick a platform which you want to install firebase to for this example we will choose IOS.



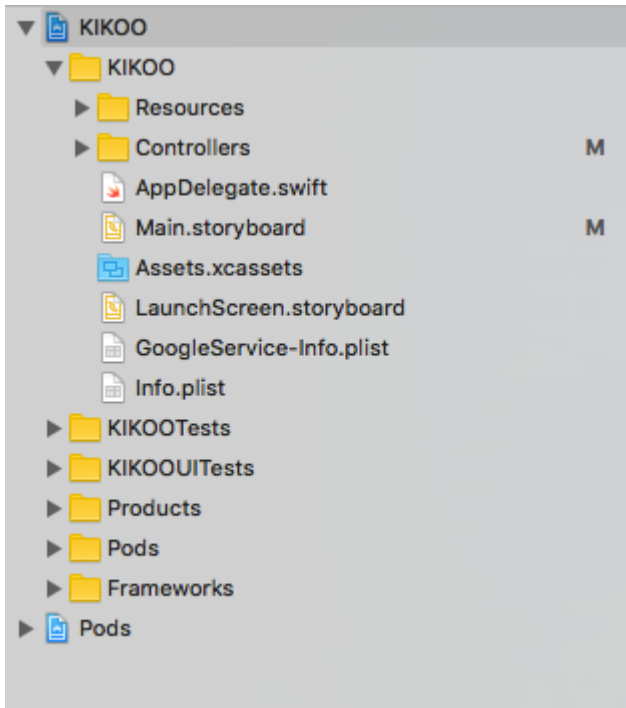
4. After selecting IOS you should see the same pop up as the one from the image below asking for the IOS Bundle and the app store id. You will only need to provide the IOS Bundle because our app isn't on the app store yet.



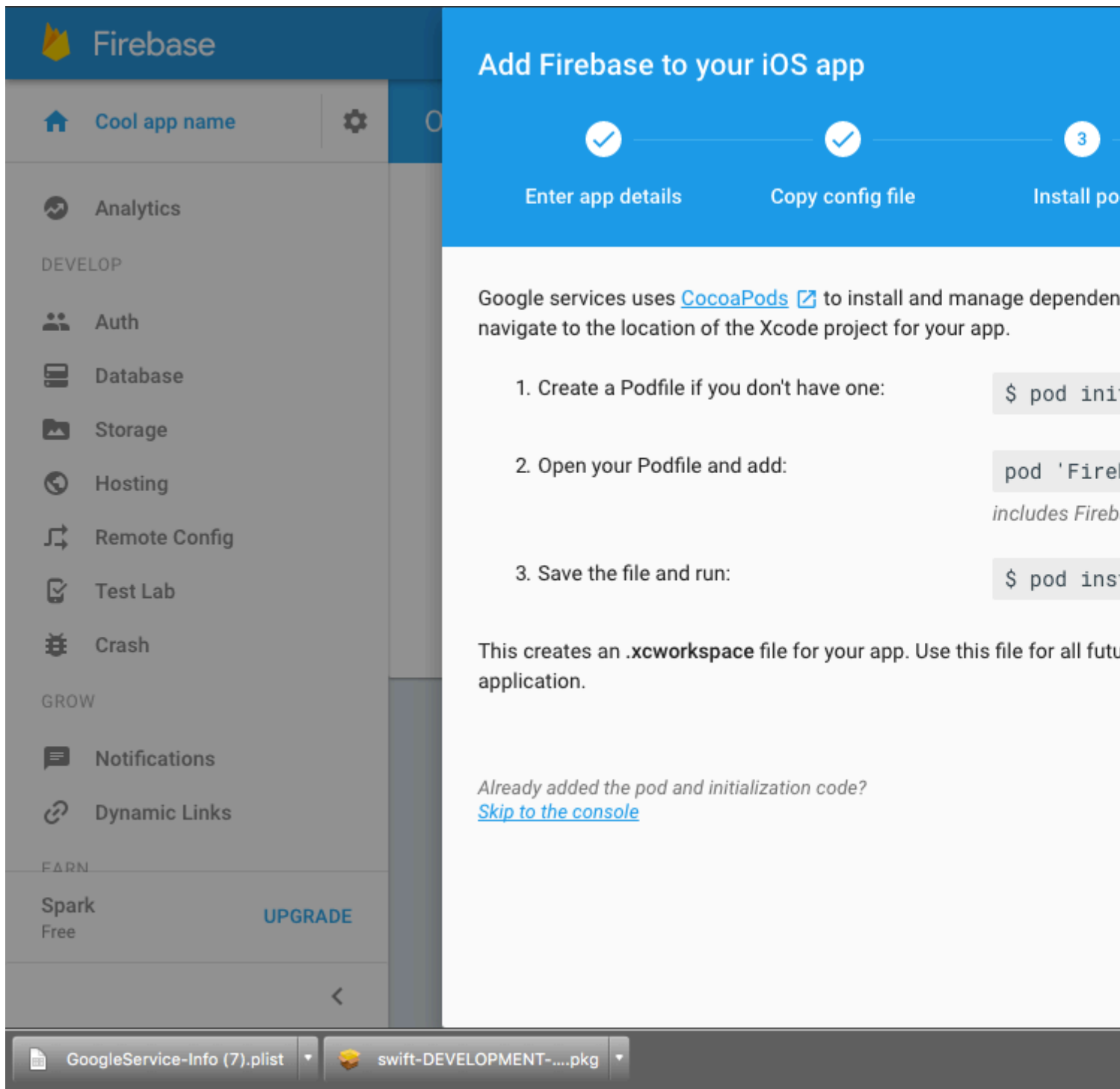
5. Get the bundle ID from xcode after creating a xcode project anyway you usually would after that you can get the bundle id for your application on the app Genral view in xcode it will be the first field at the top and once you get it paste it into the Bundle field in firebase for example mine would be 'MauginInc.KIKOO'



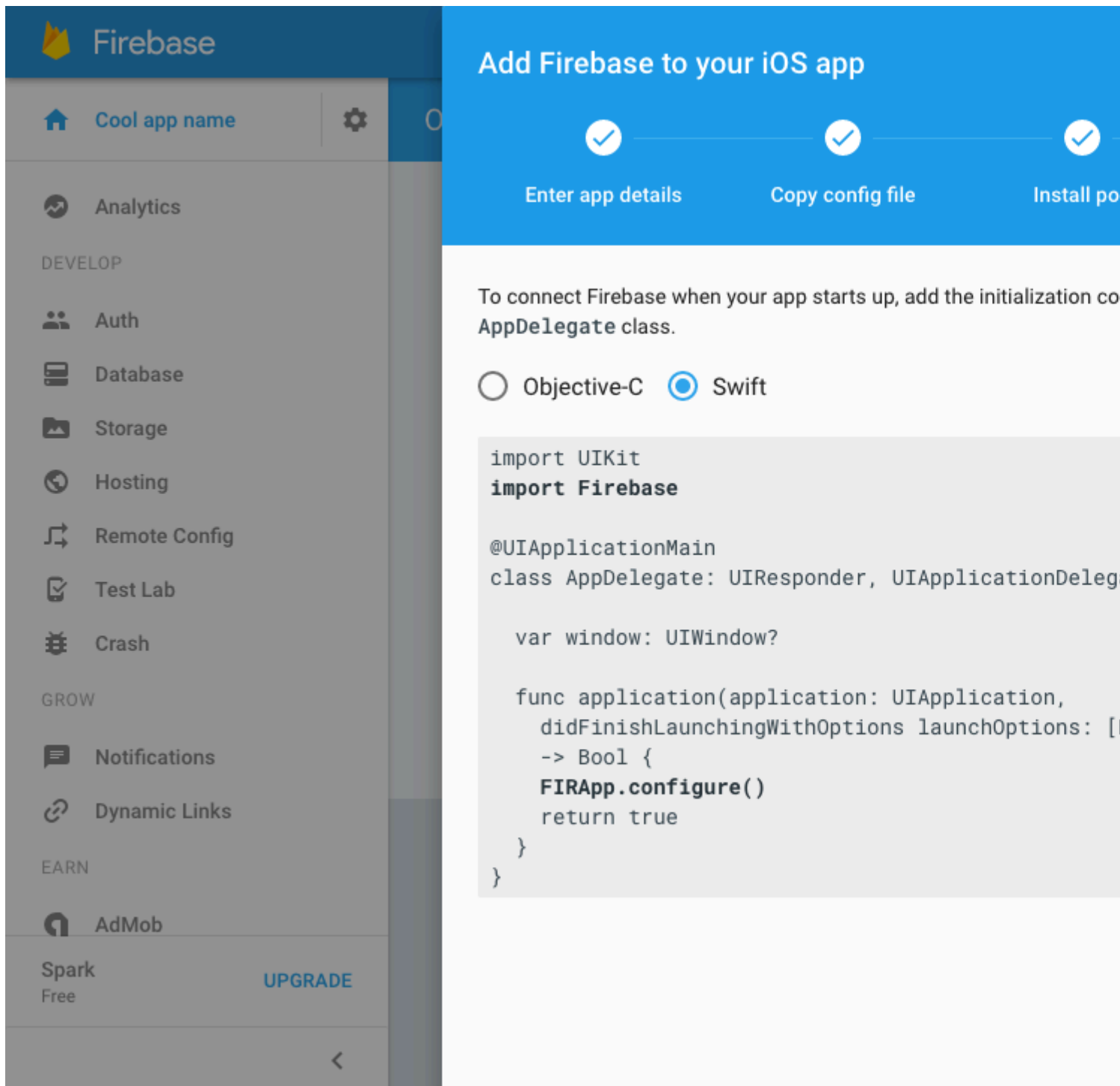
6. After you have done that and pressed 'Next' a 'GoogleService-Info.plist' file will download and what you will need to do is move that into the root folder of your app within xcode



7. You will want to initialise pods and install the firebase pods you need you can do this by going into your terminal and navigate to your xcode project folder and follow these instructions given by firebase.



8. Finally you want to configure you app to let swift do what it does best and that is making app development a whole lot more easier and efficient all you need to do is edit you AppDelegate.swift files the same the pop up shows you.



That's all you now have firebase installed in your xcode project for IOS

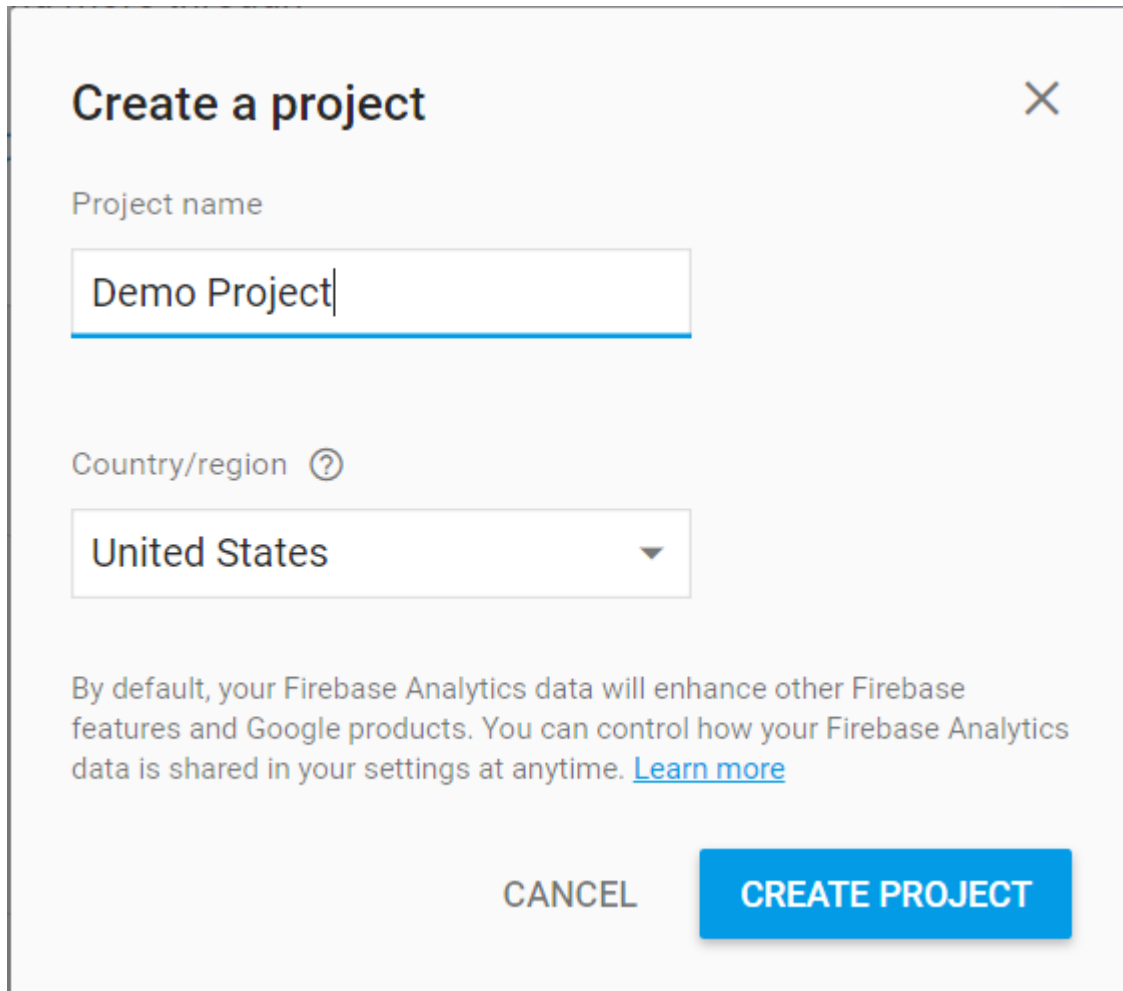
Getting started in Firebase with a simple Hello World web app in JavaScript

This example will demonstrate how to get started with Firebase in your web apps with JavaScript.

We are going to add a **text child** in our Firebase Database and display it in realtime on our web app.

Lets get started.

- Go to the Firebase Console - <https://console.firebase.google.com> and create a new project. Enter the project name, Country/region and click on **create project**.



Create a project ×

Project name

Demo Project

Country/region ?

United States ▼

By default, your Firebase Analytics data will enhance other Firebase features and Google products. You can control how your Firebase Analytics data is shared in your settings at anytime. [Learn more](#)

CANCEL CREATE PROJECT

- Now create a file **index.html** on your computer. And add the following code to it.

```
<body>
  <p>Getting started with Firebase</p>
  <h1 id="bigOne"></h1>
  <script>
    // your firebase JavaScript code here
  </script>
</body>
```

- Now go to your project on Firebase Console and you can see this

Welcome to Firebase! Get started here.



Add Firebase to
your iOS app



Add Firebase to
your Android app



Add Firebase
your web ap

- Now click on **Add Firebase to your web app**. You will the following pop up, click on copy button

Add Firebase to your web app

Copy and paste the snippet below at the bottom of your HTML, before other `script` tags

```
<script src="https://www.gstatic.com/firebasejs/3.7.4/firebase.js"></script>  
// Initialize Firebase  
var config = {  
  apiKey: "AIzaSyDjvYmJwZep_CBN_0nBbnH...",  
  authDomain: "my-app-9d48c.firebaseio.com",  
  databaseURL: "https://my-app-9d48c.firebaseio.com",  
  storageBucket: "my-app-9d48c.appspot.com",  
  messagingSenderId: "1069550000000"  
};  
firebase.initializeApp(config);</script>
```

Check these resources to learn more about Firebase for web apps:

[Get Started with Firebase for Web Apps](#) [Firebase Web SDK API Reference](#)

[Firebase Web Samples](#) 

- Now go to your **index.html** file and add the snippet to the script section as following

```
<body>

<p>Getting started with Firebase</p>
<h1 id="bigOne"></h1>

<script src="https://www.gstatic.com/firebasejs/3.7.4/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "apiKey",
    authDomain: "authDomain",
    databaseURL: "databaseURL",
    storageBucket: "storageBucket",
    messagingSenderId: "messagingSenderId"
  };
  firebase.initializeApp(config);
</script>
```

```
</body>
```

- Now you have completed adding Firebase initialization code. Now we need to get our **text** value from the database.
- To do that add the following code (Initialize Firebase already added in last step. Don't re-add) inside the script in **index.html**

```
<script>

// Initialize Firebase
var config = {
  apiKey: "apiKey",
  authDomain: "authDomain",
  databaseURL: "databaseURL",
  storageBucket: "storageBucket",
  messagingSenderId: "messagingSenderId"
};
firebase.initializeApp(config);

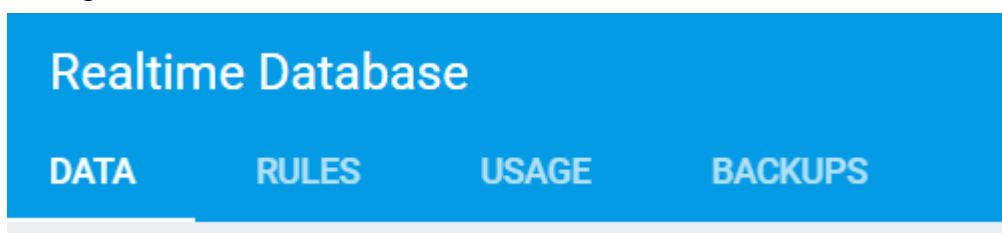
// getting the text value from the database
var bigOne = document.getElementById('bigOne');
var dbRef = firebase.database().ref().child('text');
dbRef.on('value', snap => bigOne.innerText = snap.val());

</script>
```

- Now we are all done with the **index.html** file and now let's go the **Database** in Firebase Console.
- You will see that its blank and empty right now. Lets add the a **text child** in the database and add any value to it.

A screenshot of the Firebase Database console. At the top, there is a redacted path followed by a field containing 'null' and a red 'X' icon. Below this, there is a form with two main sections: 'Name' and 'Value'. The 'Name' section has a dropdown menu currently showing 'text'. The 'Value' section has a text input field containing 'Hello Firebase' and a red 'X' icon. At the bottom of the form, there are two buttons: 'CANCEL' and 'ADD'.

- Now click on **ADD** button.
- Now go the **RULES** section in the Database.



- For development purpose right now, we will right now enable all the **read and write** queries.

```
{
  "rules": {
    ".read": "true",
    ".write": "true"
  }
}
```

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

- Now open index.html in the browser
- You will see the text value on your page as following -

Getting started with Firebase

Hello Firebase

- Now if you go back to your database and change the **text child** value to something else, you will see that the text in the browser also changes without any refresh or reload. This is how **realtime database** works on Firebase.

Read Getting started with firebase online: <https://riptutorial.com/firebase/topic/816/getting-started-with-firebase>

Chapter 2: Cloud Functions for Firebase

Introduction

Firebase launched its beta release of Cloud Functions for Firebase which is similar to using of Cloud Functions on Google Cloud Platform.

Cloud Functions is a hosted, private, and scalable Node.js environment where you can run JavaScript code. Firebase SDK for Cloud Functions integrates the Firebase platform by letting you write code that responds to events and invokes functionality exposed by other Firebase features.

Examples

Send welcome notification emails to the users for subscribing.

Use the GitHub repository to get the entire code: <https://github.com/firebase/functions-samples/blob/master/quickstarts/email-users>

- Copy or clone the repository in your computer.

Now go to your Firebase Console

- Create a Firebase Project using the Firebase Console.
- Enable the **Google** Provider in the **Auth** section.
- Paste the **Web initialization** snippet from: **Firebase Console > Overview > Add Firebase** to your web app in the **public/index.html** where the **TODO** is located.

```
* TODO(DEVELOPER): Paste the initialization snippet from: Firebase Console > Overview > Add
Firebase to your web app. *
```

```
*****
-->
<script src="https://www.gstatic.com/firebasejs/3.7.3/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "your apiKey",
    authDomain: "authDomain.firebaseio.com",
    databaseURL: "https://databaseURL.firebaseio.com",
    storageBucket: "storageBucket.appspot.com",
    messagingSenderId: "messagingID"
  };
  firebase.initializeApp(config);
</script>
```

Install Firebase CLI in your computer

- If you don't have **NodeJS** installed already, install it from <https://nodejs.org/en/> (Make sure to have the updated version of NodeJS installed on your computer.)
- Open command prompt/terminal and install it with **npm install -g firebase-tools** and then configure it with **firebase login**
- To choose your project you created now ==> Configure the CLI locally by using **firebase use --add** and select your project in the list.
- Install dependencies locally by running: **cd functions; npm install; cd -**

Set Google Cloud environment variables

- Set the **gmail.email** and **gmail.password** Google Cloud environment variables to match the email and password of the Gmail account used to send emails. For this **open the command prompt or terminal** and type the following Firebase CLI command:

```
firebase functions:config:set gmail.email="myusername@gmail.com"
gmail.password="secretpassword"
```

Deploy the project and test

- To deploy the project open the **cmd/terminal** and use the command **firebase deploy** to start the deployment.

```
=== Deploying to '...'...

i  deploying database, functions, hosting
+  database: rules ready to deploy.
i  functions: ensuring necessary APIs are enabled...
i  runtimeconfig: ensuring necessary APIs are enabled...
+  functions: all necessary APIs are enabled
+  runtimeconfig: all necessary APIs are enabled
i  functions: preparing functions directory for uploading...
i  functions: packaged functions (1.85 KB) for uploading
+  functions: functions folder uploaded successfully
i  hosting: preparing public directory for upload...
Uploading: [=====] 46%+  hosting: public folder
+  hosting: 82 files uploaded successfully
i  starting release process (may take several minutes)...
i  functions: updating function sendEmailConfirmation...
+  functions[sendEmailConfirmation]: Successful update operation.
+  functions: all functions deployed successfully!

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/.../overview
Hosting URL: https://...firebaseapp.com ← Use this URL to open
```


- Once it gets done, use the command to open the site in browser **firebase open hosting:site** or manually do it from the url displayed.

Read Cloud Functions for Firebase online: <https://riptutorial.com/firebase/topic/9580/cloud-functions-for-firebase>

Chapter 3: Crash Reporting

Remarks

Crash Reporting creates detailed reports of the errors in your app.

Errors are grouped into clusters of similar stack traces and triaged by the severity of impact on your users. In addition to automatic reports, you can log custom events to help capture the steps leading up to a crash.

Crash Reporting is currently in beta release while we resolve some known issues on Android and iOS.

Official Documentantion

<https://firebase.google.com/docs/crash/>

Examples

Setup Crash Reporting in Android

1. Complete the [Installation and setup part](#) to connect your app to Firebase.
This will create the project in Firebase.
2. Add the dependency for Firebase CrashReporting to your module-level `build.gradle` file:

```
compile 'com.google.firebase:firebase-crash:9.4.0'
```

Report the error in Android

Firebase Crash Reporting automatically generates reports for fatal errors (or uncaught exceptions).

You can create your custom report using:

```
FirebaseCrash.report(new Exception("My first Android non-fatal error"));
```

You can check in the log when FirebaseCrash initialized the module:

```
07-20 08:57:24.442 D/FirebaseCrashApiImpl: FirebaseCrash reporting API initialized
07-20 08:57:24.442 I/FirebaseCrash: FirebaseCrash reporting initialized
com.google.firebase.crash.internal.zzg@3333d325 07-20 08:57:24.442
D/FirebaseApp: Initialized class com.google.firebase.crash.FirebaseCrash.
```

And then when it sent the exception:

07-20 08:57:47.052 D/FirebaseCrashApiImpl: **throwable java.lang.Exception: My first Android non-fatal error** 07-20 08:58:18.822
D/FirebaseCrashSenderServiceImpl: **Response code: 200** 07-20 08:58:18.822
D/FirebaseCrashSenderServiceImpl: **Report sent**

You can add custom logs to your report with

```
FirebaseCrash.log("Activity created");
```

Read Crash Reporting online: <https://riptutorial.com/firebase/topic/4669/crash-reporting>

Chapter 4: Database Rules

Introduction

With Firebase Realtime Database, your Database rules is your server side security. You need to be very careful and aware of who has access to your database. It is important that no one gains access to your data that shouldn't.

By default, the Firebase Realtime Database rules allow any authenticated user to read and write all the data, this is probably not what you want your app to do.

Take a look at the below examples for different scenarios.

Remarks

The Firebase Realtime Database provides a flexible, expression-based rules language with JavaScript-like syntax to easily define how your data should be structured, how it should be indexed, and when your data can be read from and written to. Combined with our authentication services, you can define who has access to what data and protect your users' personal information from unauthorized access.

By default, your database rules require Firebase Authentication and grant full read and write permissions only to authenticated users. The default rules ensure your database isn't accessible by just anyone before you get a chance to configure i

Official Documentation

<https://firebase.google.com/docs/database/security/quickstart>

Examples

How to configure rules

1. Go in the Firebase console.
2. Choose your project
3. Click on the Database section on the left, and then select the Rules tab.

If you would like to test your security rules before putting them into production, you can simulate operations in the console using the Simulate button in the upper right of the rules editor.

The default rules

The default rules require Authentication.

They allow full read and write access to authenticated users of your app. They are useful if you

want data open to all users of your app but don't want it open to the world.

```
// These rules require authentication
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

How to set your files publicly readable and writable

Just define:

```
// These rules give anyone, even people who are not users of your app,
// read and write access to your database
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

It can be useful during development but pay attention because This level of access means **anyone can read or write to your database**.

How to disable read and write access

You can define a private rules to disable read and write access to your database by users. With these rules, **you can only access the database when you have administrative privileges (which you can get by accessing the database through the Firebase console or by [signing in from a server](#))**.

```
// These rules don't allow anyone read or write access to your database
{
  "rules": {
    ".read": false,
    ".write": false
  }
}
```

How to grant access only to authenticated users

Here's an example of a rule that gives each authenticated user a personal node at `/users/$user_id` where `$user_id` is the ID of the user obtained through **Authentication**.

```
// These rules grant access to a node matching the authenticated
// user's ID from the Firebase auth token
{
  "rules": {
    "users": {
      "$user_id": {
```

```

        ".read": "$user_id === auth.uid",
        ".write": "$user_id === auth.uid"
    }
}
}
}

```

How to allow reading specific item from group, but prevent listing group members

It is common practice to create groups of items by creating simple value nodes with item ID as key. For example, we can add a user to the group "administrators" by creating a node at `/administrators/$user_id` with a value `true`. We don't want anyone to know who administrators are, for security reasons, but we still want to check if a Authenticated user is **administrator**. With these rules we can do just that:

```

{
  "rules": {
    "administrators": {
      // No one can list administrators
      ".read": "false",
      "$uid": {
        // Authenticated user can check if they are in this group
        ".read": "$uid === auth.uid",
        // Administrators can write
        ".write": "data.parent().child(auth.uid).val() === true",
        // Allow only add or delete, no duplicates
        ".validate": "!data.exists() || !newData.exists() || newData.isBoolean()",
      }
    }
  }
}

```

Read Database Rules online: <https://riptutorial.com/firebase/topic/3352/database-rules>

Chapter 5: Email Verification after Sign Up

Syntax

- Send email verification to logged in user's email address on file. Firebase allows you to [customize what your email entails](#)
- When email hits user's email account, the user clicks on
- Using your Router of choice (used angular-ui-router in above example), intercept parameters in the URL.
- Chew the params using the `applyCode` function in Firebase.
- See below for the functions involved in the above process.

Parameters

The Function...	Does
<code>sendEmailVerification()</code>	Sends a verification email to a user.
<code>applyActionCode()</code>	Applies the action code which changes <code>emailVerified</code> from <code>false</code> to <code>true</code>

Remarks

The above pretty much sums up how to use the email verification scheme with Firebase. So far, it stands as one of the simplest ways to verify email I have seen.

There's a bit of an extended explanation of the above example available on [Email Verification with Firebase 3.0 SDK](#).

Examples

Send-cum-Process Verification Action Code - AngularJS

```
// thecontroller.js
$scope.sendVerifyEmail = function() {
  console.log('Email sent, whaaaaam!');
  currentAuth.sendEmailVerification();
}

// where currentAuth came from something like this:
// routerconfig

....
templateUrl: 'bla.html',
resolve: {
  currentAuth:['Auth', function(Auth) {
```

```

        return Auth.$requireSignIn() // this throws an AUTH_REQUIRED broadcast
    }]
}
...

// intercept the broadcast like so if you want:

....

$scope.$on("$stateChangeError", function(event, toState, toParams, fromState, fromParams,
error) {
    if (error === "AUTH_REQUIRED") {
        $state.go('login', { toWhere: toState });
    }
});
....

// So user receives the email. How do you process the `oobCode` that returns?
// You may do something like this:

// catch the url with its mode and oobCode
.state('emailVerify', {
    url: '/verify-email?mode&oobCode',
    templateUrl: 'auth/verify-email.html',
    controller: 'emailVerifyController',
    resolve: {
        currentAuth: ['Auth', function(Auth) {
            return Auth.$requireSignIn()
        }]
    }
})

// Then digest like so where each term is what they sound like:

.controller('emailVerifyController', ['$scope', '$stateParams', 'currentAuth', 'DatabaseRef',
function($scope, $stateParams, currentAuth, DatabaseRef) {
    console.log(currentAuth);
    $scope.doVerify = function() {
        firebase.auth()
            .applyActionCode($stateParams.oobCode)
            .then(function(data) {
                // change emailVerified for logged in User
                toastr.success('Verification happened', 'Success!');
            })
            .catch(function(error) {
                $scope.error = error.message;
                toastr.error(error.message, error.reason, { timeOut: 0 });
            })
    };
}
])

```

Read Email Verification after Sign Up online: <https://riptutorial.com/firebase/topic/3380/email-verification-after-sign-up>

Chapter 6: Firebase Realtime Database with Android

Examples

How to connect Realtime database with Android Application

How to implement FirebaseRealTime database in android application. Following is the steps for do it.

1. First install firebase sdk, If you dont know how to install then following is the URL for help. [Install Firebase SDK](#)
2. After that register your project in firbase console, URL of the firbase console is [Firebase Console Url](#)
3. After successfully complet above to step add following dependency in you application level gradel. compile 'com.google.firebase:firebase-database:9.2.1'
4. Also one more thing configure your firebase database rules. If you dont how to configure then following is the URL which help you. [Configure firebase Rules](#)
5. Now after all thing done, Original code is start, First retrieve your database instance throw FirebaseDatabase like following,

```
FirebaseDatabase database = FirebaseDatabase.getInstance(); DatabaseReference myRef = database.getReference("message");
```

You can now create different different object of DatabaseReference for the access different node,

6. Now you can save or retrieve data using DataBaseReference like following way, For the save :

```
myRef.setValue("Demo for Save");
```

Read data :

```
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());  
    }  
});
```

Note : This is the only introduction topic how to implement database in android application lost of more thing available in FirebaseRealtime database,

Read Firbase Realtime Database with Android online:

<https://riptutorial.com/firebase/topic/6482/firebase-realtime-database-with-android>

Chapter 7: Firebase Console

Syntax

1. Firebase Analytics Example.
2. Firebase Console Explanation for each components.

Parameters

Firebase Analytics	Firebase analytics & It's different components
Firebase Console	How it works? & How are details shown in the dashboard?

Remarks

This document is very useful for those who are the beginner of the firebase analytics. This will be very helpful to understand how's firebase analytics works in the different scenario.

Examples

Firebase All In One

[Firebase Console information in detail](#)

[Android : Firebase Analytics Example](#)

Steps For Android :

- Download code from the link
- Check FirebaseAnalyticsActivity
- That's all you will understand how's the firebase analytics works for the different scenario.

Read Firebase Console online: <https://riptutorial.com/firebase/topic/6660/firebase-console>

Chapter 8: Firebase Offline Capabilities

Introduction

In this post you will find the different ways to implement offline capabilities when using `Firebase`, information about when and why could be a good idea enable offline capabilities and examples of it with Android platform.

Remarks

What should I use? Disk persistence or `keepSynced` calls?

From my experience I can say that it always depends of what your app is working, and how you manage the transactions and database of your application. If for example you have an application where the user is just writing and reading data but he is not able to delete or edit it, use `DiskPersistence` would be the right choice.

Also, `DiskPersistence` will store data in cache, which means that your app will use more space in the user's devices, which maybe is not the best idea in your case.

In other hand, if your application manages a lot of complex transactions and your data is updated really often, possibly you should avoid `DiskPersistence` and use `keepSynced` in the references that you want to keep fresh.

Why?

`DiskPersistence` stores the data retrieved in local, which sometimes can cause lot of desynchronization showing your data if you don't use it together with continuous `ListenerValueEvents`. For example:

1. User A writes a message "Hello World" on your app, which is received for user B
2. User B download message from User A in his phone and see the message "Hello World"
3. User A edit's his message to "Firebase is cool".
4. User B will still watching "Hello World" message even if he updates the data cause the snapshot ref is the same when Firebase filter for it.

To avoid this the best idea is keep continuous listeners in the references that you want to track all time.

Can I use both together?

Of course you can, and in most of the apps possibly is the best idea to avoid download a lot of data and give the user the possibility to work with your app even if he has no connection.

If you don't care about use cache space in the user device, I recommend you to enable `diskPersistence` in your `FirebaseDatabase` object and also add a `keepSync` flags to each reference that can have a lot of times in a short space time or you want to keep fresh all time.

Examples

Enable disk persistence (Android / iOS only)

To enable disk persistence you should enable the flag `persistenceEnabled` in the `FirebaseDatabaseInstance` object of your application:

Android

```
FirebaseDatabase.getInstance().setPersistenceEnabled(true);
```

iOS

```
Database.database().isPersistenceEnabled = true //Swift  
[FIRDatabase database].persistenceEnabled = YES; //Objective-C
```

If you want to disable the persistence in some moment of your app lifecycle you should remember to disable it in the same way:

Android

```
FirebaseDatabase.getInstance().setPersistenceEnabled(false);
```

iOS

```
Database.database().isPersistenceEnabled = false //Swift  
[FIRDatabase database].persistenceEnabled = NO; //Objective-C
```

Keeping data fresh (Android/iOs Only)

Firebase synchronizes and stores a local copy of the data for active listeners when used on mobile devices. In addition, you can keep specific locations in sync.

Android :

```
DatabaseReference workoutsRef = FirebaseDatabase.getInstance().getReference("workouts");  
scoresRef.keepSynced(true);
```

iOs:

```
//Objective-c  
FIRDatabaseReference *scoresRef = [[FIRDatabase database] referenceWithPath:@"scores"];  
[scoresRef keepSynced:YES];  
//Swift
```

```
let scoresRef = Database.database().reference(withPath: "scores")
scoresRef.keepSynced(true)
```

Firebase client automatically downloads the data at these locations and keeps it updated even if the reference has no active listeners. You disable synchronization with the following line of code.

Android :

```
scoresRef.keepSynced(false);
```

iOS:

```
[scoresRef keepSynced:NO]; //Objective-C
scoresRef.keepSynced(false) //Swift
```

Read **Firebase Offline Capabilities** online: <https://riptutorial.com/firebase/topic/10777/firebase-offline-capabilities>

Chapter 9: Firebase Queue

Examples

How to use firebase queue as a backend for your application

Firebase provides backend as a service, as application developer you do not have an option to have backend code.

This example shows how using firebase queue, create backend which will operate on the top of firebase database and serve as a backend for your frontend application.

Before getting into the code let's understand the architecture, how it will work. For brevity let's suppose that we are using web site as a frontend and NodeJs server as a backend

Prerequisites

1. Create firebase application using your google account
2. Add firebase to your web page. Use `bower install firebase --save`
3. Create service account using your new created firebase account (Settings->Permissions -> Service Accounts -> CREATE SERVICE ACCOUNT -> (specify name and check this "Furnish a new private key" checkbox) -> save the json file, we will need that later.
4. Configure NodeJs server which can be hosted in your preferred environment
5. Create following endpoint inside `queue/specs`

"request_response":

```
{
  "error_state": "request_error_processing",
  "finished_state": "finished_state",
  "in_progress_state": "request_in_progress",
  "start_state": "request_started"
}
```

6. Inside NodeJs server install firebase server side version, `npm install firebase --save`, and initialize your service account using the json file which we got from the step 3, it looks like this

```
firebase.initializeApp({ serviceAccount: './your_file.json', databaseURL:
'get_from_firebase_account' });
```

Architecture

Here is the whole cycle how it works.

On the frontend side you gonna do these steps

1. Using firebase web sdk you are writing your requests directly into firebase database in the endpoint 'queue/tasks', lets call that your request which you are sending to the backend.
2. after inserting your task you are registering listener on the endpoint `queue/tasks/{taskKey}` which would be called when backend finishes processing your request, writing response inside above task

In the backend side you gonna do these steps

1. Create server which infinitely listens endpoint 'queue/tasks'
2. Processes your tasks and writing back response data inside `queue/tasks/response`
3. Remove the task

First of all create this helper function, which provides a way of handling callbacks and promises together

```
function createPromiseCallback() {
  var cb;
  var promise = new Promise(function (resolve, reject) {
    cb = function (err, data) {
      if (err) return reject(err);
      return resolve(data);
    };
  });
  cb.promise = promise;
  return cb;
}
```

In the frontend side you gonna have this function

```
function sendRequest(kind, params, cb) {

  cb = cb || createPromiseCallback();
  var requestObject = {
    kind: kind,
    params: params
  };
  var tasksRef = firebase.database().ref('queue/tasks');

  var requestKey = tasksRef.push().key;

  var requestRef = tasksRef.child(requestKey);

  function requestHandshake(snap) {
    if (snap && snap.exists() && (snap.val().response || snap.val()._state ===
config.firebase.task.finishState || snap.val()._error_details)) {
      var snapVal = snap.val();
      if (snapVal._error_details) {
        cb(snapVal._error_details.error);
      } else {
        cb(null, snapVal.response);
      }
    }
    requestRef.off('value', requestHandshake);
  }
}
```



```

var bulkUpdate = {};
bulkUpdate['queue/tasks/' + requestKey + '/request'] = requestObject;
bulkUpdate['queue/tasks/' + requestKey + '/_state'] = config.firebase.task.startState;

firebase.database().ref().update(bulkUpdate)
  .then(function (snap) {
    requestRef.on('value', requestHandshake);
  }).catch(function (err) {
    cb(err);
  });

return cb.promise;
}

```

you can use this function like `sendRequest('CreateHouseFacade', {houseName:'Test'})`.

Kind parameter is for backend, to know what method to call for processing request. Params is for passing additional parameter information.

And here is the backend code

```

const database = firebase.database();
const queueRef = database.ref('queue');

const queueOptions = {
  'specId': 'request_response',
  'sanitize': false,
  'suppressStack': false,
  'numWorkers': 3
};

function removeTask(task) {
  var taskRef = queueRef.child(`tasks/${task._id}`);
  return taskRef.remove();
}

function processTask(data, progress, resolve, reject) {
  try {
    requestHandler(data.request).then(response => {
      data.response = response || null;
      return resolve(data);
    }).catch(err => {
      return reject(err);
    }).then(snap => {
      removeTask(data);
    });
  } catch (err) {
    reject(err).then(snap => removeTask(data));
  }
}

function requestHandler(request) {
  if (!request || !request.kind) throw new Error('Absent Request or Kind');
  var deferredResponse = requestHandlerFactory(request.kind, request.params);
  return deferredResponse;
}

function requestHandlerFactory(kind, params) {

```

```
// It includes mapping all your backend services
switch (kind) {
  case 'CreateHouseFacade': return myService(params)
  default: throw new Error(`Invalid kind ${kind} was specified`);
}
```

The function `myService` contains your business logic code which gonna accomplishing `CreateHouseFacade` request.

Read Firebase Queue online: <https://riptutorial.com/firebase/topic/7619/firebase-queue>

Chapter 10: FirebaseUI

Remarks

Firebase is a suite of integrated products designed to help you develop your application, grow an engaged user base, and earn more money. It includes tools that help you build your app, such as a realtime database, file storage, and user authentication, as well as tools to help you grow and monetize your app, such as push notifications, analytics, crash reporting, and dynamic links.

You can think of Firebase as a set of Lego bricks that you can use to build your masterpiece. Just like bricks, Firebase is relatively unopinionated, since there are an infinite number of ways to combine the pieces and we're not going to tell you that certain ways are wrong :)

FirebaseUI is built on Firebase and provides developers simple, customizable, and production-ready native mobile bindings on top of Firebase primitives to eliminate boilerplate code and promote Google best practices

In the Lego analogy, FirebaseUI is a set of pre-built kits with instructions that you can take off the shelf and tweak to suit your needs. You can see how we used the individual components of Firebase to build FirebaseUI because FirebaseUI is open source. FirebaseUI has to be opinionated--we're telling you how we think the bricks should go together, so we make some choices. But because FirebaseUI is open source, you can go in and change what we're doing to better suit your individual needs.

If you're building a Lego city, you'd rather pull a bunch of houses from a pre-build collection and modify slightly to suit your needs than start from scratch and design each building by hand, right?

FirebaseUI let's you do exactly this, which is why we include it in our sample apps and examples. Developers (ourselves included) are lazy--we want the best reuse of our code and the most concise examples, and FirebaseUI allows us to provide really high quality examples that translate to really good user experiences at a fraction of the development cost.

Examples

Getting Started with FirebaseUI

FirebaseUI offers [Android](#), [iOS](#), and [Web](#) clients. You can get started with them like so:

Android:

```
// app/build.gradle

dependencies {
    // Single target that includes all FirebaseUI libraries
    compile 'com.firebaseui:firebase-ui:0.5.2'

    // FirebaseUI Database only
```

```
compile 'com.firebaseui:firebase-ui-database:0.5.2'

// FirebaseUI Auth only
compile 'com.firebaseui:firebase-ui-auth:0.5.2'
}
```

iOS:

```
# Podfile

# Pull in all Firebase UI features
pod 'FirebaseUI', '~> 0.5'

# Only pull in the "Database" FirebaseUI features
pod 'FirebaseUI/Database', '~> 0.5'

# Only pull in the "Auth" FirebaseUI features (including Facebook and Google)
pod 'FirebaseUI/Auth', '~> 0.5'

# Only pull in the "Facebook" login features
pod 'FirebaseUI/Facebook', '~> 0.5'

# Only pull in the "Google" login features
pod 'FirebaseUI/Google', '~> 0.5'
```

Web:

```
<!--Include FirebaseUI sources in HTML-->

<script src="https://www.gstatic.com/firebasejs/ui/live/0.5/firebase-ui-auth.js"></script>
<link type="text/css" rel="stylesheet"
href="https://www.gstatic.com/firebasejs/ui/live/0.5/firebase-ui-auth.css" />
```

Read FirebaseUI online: <https://riptutorial.com/firebase/topic/6418/firebaseui>

Chapter 11: FirebaseUI (Android)

Examples

Adding the dependencies

FirebaseUI is just an open-source library by Google that provides easy UI bindings for Firebase Auth and Firebase Database.

To begin adding FirebaseUI to your app, add these dependencies in your app's `build.gradle` file:

```
android {
    // ...
}

dependencies {
    // Required for FirebaseUI Database
    compile 'com.google.firebase:firebase-database:9.4.0'
    compile 'com.firebaseui:firebase-ui-database:0.5.1'

    // FirebaseUI Auth only
    compile 'com.google.firebase:firebase-auth:9.4.0'
    compile 'com.firebaseui:firebase-ui-auth:0.5.1'

    // Single dependency if you're using both
    compile 'com.firebaseui:firebase-ui:0.5.1'
}

apply plugin: 'com.google.gms.google-services'
```

Populating a ListView

Assuming you have already set up an app in Android Studio, add a `ListView` to a layout (or skip if that's already done):

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Your toolbar, etc -->

    <ListView
        android:id="@+id/list_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</android.support.design.widget.CoordinatorLayout>
```

Now let's create a model for the data we're going to populate our `ListView` with:

```

public class Person {

    private String name

    public Person() {
        // Constructor required for Firebase Database
    }

    public String getName() {
        return name;
    }

}

```

Make sure your `ListView` has an id, then create a reference to it in your `Activity` and set its adapter to a new `FirestoreListAdapter`:

```

public class MainActivity extends AppCompatActivity {

    // ...

    private ListView mListView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Find the ListView
        mListView = (ListView) findViewById(R.id.list_view);

        /*
         * Create a DatabaseReference to the data; works with standard DatabaseReference
         methods
         * like limitToLast() and etc.
         */
        DatabaseReference peopleReference = FirebaseDatabase.getInstance().getReference()
            .child("people");

        // Now set the adapter with a given layout
        mListView.setAdapter(new FirestoreListAdapter<Person>(this, Person.class,
            android.R.layout.one_line_list_item, peopleReference) {

            // Populate view as needed
            @Override
            protected void populateView(View view, Person person, int position) {
                ((TextView) view.findViewById(android.R.id.text1)).setText(person.getName());
            }
        });
    }
}

```

After you've done that, add some data to your database and watch the `ListView` populate itself.

Read **FirestoreUI (Android)** online: <https://riptutorial.com/firebase/topic/6610/firebaseui--android->

Chapter 12: How do I listen for errors when accessing the database?

Introduction

There are many reasons a read or write operation may fail. A frequent one is because your security rules reject the operation, for example because you're not authenticated (by default a database can only be accessed by an authenticated user) or because you're writing/listening at a location where you don't have permission.

Examples

Detect errors when writing a value on Android

There are many reasons a write operation may fail. A frequent one is because your security rules reject the operation, for example because you're not authenticated (by default a database can only be accessed by an authenticated user).

You can see these security rule violations in the logcat output. But it's easy to overlook these. You can also handle them in your own code and make them more prominently visible, which is especially useful during development (since your JSON, rules and code change often).

To detect a failed write on Android you [attach a completion callback to `setValue`](#):

```
ref.setValue("My new value", new DatabaseReference.CompletionListener() {
    public void onComplete(DatabaseError databaseError, DatabaseReference databaseReference) {
        throw databaseError.toException();
    }
});
```

Throwing an exception like this ensures that it will be very difficult to overlook such an error next time.

Detect errors when reading data on Android

A frequent reason why your read operation may not work is because your security rules reject the operation, for example because you're not authenticated (by default a database can only be accessed by an authenticated user).

You can see these security rule violations in the logcat output. But it's easy to overlook these. You can also handle them in your own code and make them more prominently visible, which is especially useful during development (since your JSON, rules and code change often).

To detect a failed read on Android you must implement the `onCancelled` method of your `ChildEventListener`:

```
databaseRef.addChildEventListener(new ChildEventListener() {
    public void onChildAdded(DataSnapshot dataSnapshot, String s) { ... }
    public void onChildChanged(DataSnapshot dataSnapshot, String s) { ... }
    public void onChildRemoved(DataSnapshot dataSnapshot) { ... }
    public void onChildMoved(DataSnapshot dataSnapshot, String s) { ... }
    public void onCancelled(DatabaseError databaseError) {
        throw databaseError.toException();
    }
});
```

Or if you have a `ValueEventListener`:

```
databaseRef.addValueEventListener(new ValueEventListener() {
    public void onDataChange(DataSnapshot dataSnapshot, String s) { ... }
    public void onCancelled(DatabaseError databaseError) {
        throw databaseError.toException();
    }
});
```

With this code in place it will be pretty hard to overlook a security error when reading data on Android.

Detect errors when writing a value on iOS

There are many reasons a write operation may fail. A frequent one is because your security rules reject the operation, for example because you're not authenticated (by default a database can only be accessed by an authenticated user).

You can see these security rule violations in the output of your program. But it's easy to overlook these. You can also handle them in your own code and make them more prominently visible, which is especially useful during development (since your JSON, rules and code change often).

To detect a failed write on iOS you [attach a completion block to `setValue`](#):

```
let message = ["name": "puf", "text": "Hello from iOS"]
ref!.childByAutoId().setValue(message) { (error) in
    print("Error while writing message \(error)")
}
```

Throwing an exception like this ensures that it will be very difficult to overlook such an error next time.

Detecting errors when reading data in JavaScript

A frequent reason why your read operation may not work is because your security rules reject the operation, for example because you're not authenticated (by default a database can only be accessed by an authenticated user).

You can see these security rule violations in the JavaScript console of your browser. But it's easy to overlook these. You can also handle them in your own code and make them more prominently visible, which is especially useful during development (since your JSON, rules and code change

often).

To detect a failed read in JavaScript you must implement add a second callback to your `on()` clause:

```
ref.on('value', function(snapshot) {
    console.log(snapshot.key, snapshot.val());
}, function(error) {
    alert(error);
})
```

With this code in place it will be pretty hard to overlook a security error when reading data in JavaScript.

Detecting errors when writing a value in JavaScript

There are many reasons a write operation may fail. A frequent one is because your security rules reject the operation, for example because you're not authenticated (by default a database can only be accessed by an authenticated user).

You can see these security rule violations in the console output. But it's easy to overlook these. You can also handle them in your own code and make them more prominently visible, which is especially useful during development (since your JSON, rules and code change often).

To detect a failed write in JavaScript you attach a completion callback to `set`:

```
ref.set("My new value").catch(function(error)
    console.error(error);
    alert(error);
});
```

Showing an alert like this ensures that it will be very difficult to overlook such an error next time.

Detect errors when reading data on iOS

A frequent reason why your read operation may not work is because your security rules reject the operation, for example because you're not authenticated (by default a database can only be accessed by an authenticated user).

You can see these security rule violations in the Console output. But it's easy to overlook these. You can also handle them in your own code and make them more prominently visible, which is especially useful during development (since your JSON, rules and code change often).

To detect a failed read on iOS you must implement the `withCancel` block of your observer:

```
ref!.child("notAllowed").observe(.value, with: { (snapshot) in
    print("Got non-existing value: \(snapshot.key)")
}, withCancel: { (error) in
    print(error)
})
```

Read [How do I listen for errors when accessing the database?](https://riptutorial.com/firebase/topic/5548/how-do-i-listen-for-errors-when-accessing-the-database-) online:

<https://riptutorial.com/firebase/topic/5548/how-do-i-listen-for-errors-when-accessing-the-database->

Chapter 13: How to get push key value from Firebase Database?

Introduction

In Firebase Database everything is a node, that follows the pattern key: value. Firebase Database provides us with a simple way to generate unique keys. Unique keys create new items while uploading data to a previously stored key will update.

Examples

Android Example

Let's assume we have a Dogs application, then our model will be a Dog class.

```
DatabaseReference reference = FirebaseDatabase.getInstance().getReference().child("dogs");
```

This is how to send a Dog to the database, a new unique dog and set the dog with the key.

```
String key = reference.push().getKey();
Dog dog = new Dog("Spike");
dog.setKey(key);
reference.child(key).setValue(dog);
```

The `reference.child(key).setValue(dog);` is equivalent of `reference.push().setValue(dog);` And add the benefit of getting the key inside the `Dog` object.

Read [How to get push key value from Firebase Database?](https://riptutorial.com/firebase/topic/10839/how-to-get-push-key-value-from-firebase-database-) online:

<https://riptutorial.com/firebase/topic/10839/how-to-get-push-key-value-from-firebase-database->

Chapter 14: How to use FirebaseRecyclerAdapter instead of RecyclerViewAdapter?

Examples

Here is the Example for Use FirebaseUi component FirebaseRecyclerAdapter

Hello friends before start code we have need to declare dependency for access firebase ui component, so here is the dependency which you can put it in your gradel other wise you can add dependency as jar also.

```
compile 'com.firebaseui:firebase-ui-database:0.4.0'
```

Then after we are querying in firebase database for data like following way

```
DatabaseReference databaseReference = database.getReference().child("users");
Query query = databaseReference.limitToFirst(50);
```

Then after we pass query inside of FirebaseRecyclerAdapter like following way

```
private void setUpFirebaseAdapter(Query query) {

    mFirebaseAdapter = new FirebaseRecyclerAdapter<UserModel, FirebaseUserViewHolder>
        (UserModel.class, R.layout.row_user_list, FirebaseUserViewHolder.class, query)
    {
        @Override
        protected void populateViewHolder(FirebaseUserViewHolder viewHolder, UserModel
model, int position) {
            customeLoaderDialog.hide();
            viewHolder.bindUser(model);
        }
    };

    my_recycler_view.setHasFixedSize(true);
    my_recycler_view.setLayoutManager(new LinearLayoutManager(this));
    my_recycler_view.setAdapter(mFirebaseAdapter);

}
```

ChatUserModel.java (Model Class)

```
public class ChatUserModel {
    private long badge;
    private String chat_id;
    private String isDelete;
    private String latestactivity;
    private double timestamp;
```

```

private String user_id;
private String profilePic;
private String displayName;
private boolean isGroup;
String groupId;
private String creatorId;

public String getGroupId() {
    return groupId;
}

public void setGroupId(String groupId) {
    this.groupId = groupId;
}

public String getCreatorId() {
    return creatorId;
}

public void setCreatorId(String creatorId) {
    this.creatorId = creatorId;
}

public boolean isGroup() {
    return isGroup;
}

public void setGroup(boolean group) {
    isGroup = group;
}

public ChatUserModel() {
}

public long getBadge() {
    return badge;
}

public void setBadge(long badge) {
    this.badge = badge;
}

public String getChat_id() {
    return chat_id;
}

public void setChat_id(String chat_id) {
    this.chat_id = chat_id;
}

public String getIsDelete() {
    return isDelete;
}

public void setIsDelete(String isDelete) {
    this.isDelete = isDelete;
}

public String getLatestactivity() {

```

```

        return latestactivity;
    }

    public void setLatestactivity(String latestactivity) {
        this.latestactivity = latestactivity;
    }

    public double getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(double timestamp) {
        this.timestamp = timestamp;
    }

    public String getUser_id() {
        return user_id;
    }

    public void setUser_id(String user_id) {
        this.user_id = user_id;
    }

    public String getProfilePic() {
        return profilePic;
    }

    public void setProfilePic(String profilePic) {
        this.profilePic = profilePic;
    }

    public String getDisplayName() {
        return displayName;
    }

    public void setDisplayName(String displayName) {
        this.displayName = displayName;
    }
}}

```

FirestoreChatUserViewHolder.java (RecyclerView.ViewHolder)

```

public class FirestoreChatUserViewHolder extends RecyclerView.ViewHolder implements
View.OnClickListener {

    private static final int MAX_WIDTH = 200;
    private static final int MAX_HEIGHT = 200;
    View mView;
    Context mContext;
    ChatUserModel userModel;

    public FirestoreChatUserViewHolder(View itemView) {
        super(itemView);
        mView = itemView;
        mContext = itemView.getContext();
        itemView.setOnClickListener(this);
    }

    public void bindUser(ChatUserModel userModel) {
        this.userModel = userModel;
        ImageView imgUser = (ImageView) mView.findViewById(R.id.imgUser);
    }
}

```

```

        TextView tvName = (TextView) mView.findViewById(R.id.tvName);
        TextView tvStatus = (TextView) mView.findViewById(R.id.tvStatus);
        BadgeView badgeChat = (BadgeView) mView.findViewById(R.id.badgeChat);
        if (userModel.isGroup()) {
            //
imgUser.setImageDrawable(mContext.getResources().getDrawable(R.drawable.create_group));
        } else {
            Picasso.with(mContext)
                    .load(userModel.getProfilePic())
                    .resize(MAX_WIDTH, MAX_HEIGHT)
                    .centerCrop()
                    .into(imgUser);
        }

        tvName.setText(userModel.getDisplayName());
        tvStatus.setText(userModel.getLatestactivity());
        if (userModel.getBadge() > 0) {
            badgeChat.setVisibility(View.VISIBLE);
            badgeChat.setText("" + userModel.getBadge());
        } else {
            badgeChat.setVisibility(View.GONE);
        }

    }

    @Override
    public void onClick(View view) {
        if (!userModel.isGroup()) {
            Intent intent = new Intent(mContext, ChatConverstion.class);
            intent.putExtra("chat_id", "" + userModel.getChat_id());
            intent.putExtra("reciverUserName", "" + userModel.getDisplayName());
            intent.putExtra("reciverProfilePic", "" + userModel.getProfilePic());
            intent.putExtra("reciverUid", "" + userModel.getUser_id());
            mContext.startActivity(intent);
        }
    }
}

```

row_user_list.xml (layout for row in recycler view)

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@android:color/white"
    android:orientation="horizontal"
    >

    <LinearLayout
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:id="@+id/llMainChat"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:paddingTop="@dimen/margin_small"
        android:paddingLeft="@dimen/margin_small"
        android:paddingBottom="@dimen/margin_small"
        android:paddingRight="@dimen/margin_small"
        >

```

```

<com.tristate.firebasechat.custome_view.CircleImageView
    android:id="@+id/imgUser"
    android:layout_width="@dimen/tab_top_height"
    android:layout_height="@dimen/tab_top_height"

    app:civ_border_color="@color/dark_white"
    app:civ_border_width="2dp" />

<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginLeft="@dimen/margin_medium"
>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_toLeftOf="@+id/badgeChat"
        android:layout_toStartOf="@+id/badgeChat"
        android:id="@+id/linearLayout">

            <TextView
                android:id="@+id/tvName"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:ellipsize="marquee"
                android:singleLine="true"
                android:text="Dhaval Solanki"
                android:textSize="@dimen/textsize_middle" />

            <TextView
                android:id="@+id/tvStatus"
                android:ems="3"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="center_vertical"
                android:lines="1"
                android:text="Online"
                android:textColor="@color/greenStatusBar"
                android:textSize="@dimen/textsize_small" />
        </LinearLayout>
    <com.tristate.firebasechat.custome_view.BadgeView
        android:id="@+id/badgeChat"
        android:layout_width="@dimen/margin_very_big"
        android:layout_height="@dimen/margin_very_big"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true"
        android:background="@drawable/badge_bg"
        android:gravity="center"
        android:padding="@dimen/corner_radius"
        android:text="999"
        android:textColor="@color/white"
        android:textSize="@dimen/textsize_verysmall"
        android:visibility="gone" />

</RelativeLayout>

```



```
</LinearLayout>
<View
    android:layout_alignBottom="@id/llMainChat"
    android:layout_marginTop="@dimen/margin_small"
    android:layout_marginLeft="@dimen/margin_small"
    android:layout_marginRight="@dimen/margin_small"
    android:layout_width="match_parent"
    android:background="@color/avatar_back_color"
    android:layout_height="1dp"
></View>
</RelativeLayout>
```

Read [How to use FirebaseRecyclerAdapter instead of RecyclerView? online:](https://riptutorial.com/firebase/topic/8982/how-to-use-firebase-recycleradapter-instead-of-recycleradapter-)
<https://riptutorial.com/firebase/topic/8982/how-to-use-firebase-recycleradapter-instead-of-recycleradapter->

Chapter 15: How to use the Firebase Database to keep a list of Firebase Authentication users

Examples

How to save user profile data

Every authenticated user has a Firebase `uid` that's unique across all providers and is returned in the result of every authentication method.

A good way to store your user's data is to create a node to keep all the users's data and to protect it using your security rules

- Database

```
{
  "users": {
    "uid1" : {
      "name": "Steve",
      "surname": "Jobs"
    },
    "uid2" : {
      "name": "Bill",
      "surname": "Gates"
    }
  }
}
```

- Security

```
{
  "rules": {
    "users": {
      "$uid": {
        // If node's key matches the id of the auth user
        ".write": "$uid == auth.uid"
      }
    }
  }
}
```

The `$uid` in the above rules is a so-called "dollar variable", which ensures that the rules under it are applied to all child nodes of `users`. For more information see the documentation on [Using \\$ Variables to Capture Path Segments](#).

Why save user data in the database

[Firebase Authentication](#) allows the users of your app to sign-in with social providers or their email+password. But what if you want to store additional information about a user, beyond what Firebase Authentication allows you to specify?

Or what if you want to display a list of the users in your app? Firebase Authentication doesn't have an API for this.

Most developers solve this problem by storing the additional information in a separate database. This topic covers how to store such information in the [Firebase Realtime Database](#).

Handling User Account Data in the Realtime Database

The Firebase auth system is the source of a users `uid`, `displayName`, `photoURL`, and maybe `email`. Password based accounts set these *persistent* values in the auth system via the `.updateProfile` method. Storing these values in the Realtime Database, `rDB`, `users` node poses the issue of stale data. Display names, for example, may change. To keep these values in synch use [local storage](#) in concert with `.onAuthStateChanged`.

on every `.onAuthStateChanged`

- `getItem('displayName')` and `getItem('photoURL')`
- compare to `user.displayName` and `user.photoURL`
- if different
 - `setItem('displayName')` and `setItem('photoURL')`
 - `db.ref.child('users').update` the values of `displayName` and/or `photoURL`

`.onAuthStateChanged` fires on every page load or reload, as well as on every auth state change. It potentially fires often, e.g. multi page apps. However reading and writing to local storage is synchronous and very fast so there will be no noticeable impact on app performance.

Read [How to use the Firebase Database to keep a list of Firebase Authentication users online](https://riptutorial.com/firebase/topic/1729/how-to-use-the-firebase-database-to-keep-a-list-of-firebase-authentication-users):
<https://riptutorial.com/firebase/topic/1729/how-to-use-the-firebase-database-to-keep-a-list-of-firebase-authentication-users>

Chapter 16: Push notification from custom server

Introduction

This can be done using 2 methods with **HTTP Post request**, With **Firestore admin SDK** running on your server. Here I will discuss both of them.

Examples

Firestore Cloud Messaging HTTP Protocol

From your server request to the the link below to send the notification with some request parameters

```
https://fcm.googleapis.com/fcm/send
```

While requesting add headers as follows

```
Authorization    key=<Your_key_from_the_console>
Content-Type     application/json
```

The body of the request varies

```
{
  "to" : <tokens or the topic>,
  "notification" : {
    "title":"This is a test title",
    "body":"This is the body"
  },
  "data": {
    //whatever key value payer you need to send
  }
}
```

The to parameters takes Array of tokens like

```
["token1", "token2", .....]
```

or a single token like

```
"token"
```

or a topic name starting with **/topic/** like

```
"/topic_name/"
```

For multiple topic use conditions using || and && operators like

```
"/topic_name/ && /topic2/"
```

Using Admin SDK(Node js)

At first initialize the firebase sdk and admin SDK

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');

admin.initializeApp({
  credential: admin.credential.cert({
    //your admin credential certificate generated from the console. Follow this [link][1].
  }),
  databaseURL: "https://<PROJECT_NAME>.firebaseio.com"
});
```

Create a payload JSON string as in the first example.

```
var payload = {
  notification: {
    title: "Title of the notification",
    body: "Body of the notification",
  },
  data: {
    //required key value pair
  }
};
```

Then call different send methods to send the notification.

For Topic

```
admin.messaging().sendToTopic("/topic/", payload)
  .then(function(response) {
    console.log("Successfully sent message:", response);
  })
  .catch(function(error) {
    console.log("Error sending message:", error);
  });
```

For device

```
admin.messaging().sendToDevice(token, payload).then(response=>{
  response.results.forEach((result, index) => {
    const error = result.error;
    if (error) {
      console.error('Failure sending notification to', tokens, error);
    } else{
```

```
        console.log('Sucessfully sent to '+tokens);  
    }  
});
```

Read Push notification from custom server online:

<https://riptutorial.com/firebase/topic/10548/push-notification-from-custom-server>

Chapter 17: Storage

Remarks

Firebase Storage provides secure file uploads and downloads for your Firebase apps, regardless of network quality. You can use it to store images, audio, video, or other user-generated content. Firebase Storage is backed by Google Cloud Storage, a powerful, simple, and cost-effective object storage service.

Firebase Storage stores your files in a Google Cloud Storage bucket shared with the default Google App Engine app, making them accessible through both Firebase and Google Cloud APIs. This allows you the flexibility to upload and download files from mobile clients via Firebase and do server-side processing such as image filtering or video transcoding using Google Cloud Platform. Firebase Storage scales automatically, meaning that there's no need to migrate from Firebase Storage to Google Cloud Storage or any other provider.

This integration makes files accessible directly from the Google Cloud Storage gcloud client libraries, so you can use Firebase Storage with your favorite server-side languages. For more control, you can also use the Google Cloud Storage XML and JSON APIs.

Firebase Storage integrates seamlessly with Firebase Authentication to identify users, and provides a declarative security language that lets you set access controls on individual files or groups of files, so you can make files as public or private as you want.

See the [public docs for Firebase Storage](#) for the most up to date APIs, samples, and example apps.

Examples

Getting started on iOS

Prerequisites

1. Create a new project and add an iOS app to that project in the [Firebase Console](#).
2. Download and include `GoogleServices-Info.plist` in your application.

Add Firebase Storage to your app

Add the following dependency to your project's `Podfile`:

```
pod 'Firebase/Storage'
```

Run `pod install` and open the created `.xcworkspace` file.

Follow these instructions to install Firebase without CocoaPods

Set up Firebase Storage

You must initialize Firebase before any Firebase app reference is created or used. If you have already done this for another Firebase feature, you can skip the following two steps.

Import the Firebase module:

```
// Obj-C
@import Firebase;
```

```
// Swift
import Firebase
```

Configure a `FIRApp` shared instance, typically in your application's `application:didFinishLaunchingWithOptions:` method:

```
// Obj-C
[FIRApp configure];
```

```
// Swift
FIRApp.configure()
```

Get a reference to the storage service, using the default Firebase App:

```
// Obj-C
FIRStorage *storage = [FIRStorage storage];
```

```
// Swift
let storage = FIRStorage.storage()
```

Create a reference to a file in Firebase Storage:

```
// Obj-C
FIRStorageReference *reference = [[storage reference] child:@"path/to/file.txt"];
```

```
// Swift
let reference = storage.reference().child("path/to/file.txt")
```

Upload a file to Firebase Storage:

```
// Obj-C
NSData *data = ...
FIRStorageUploadTask *uploadTask = [riversRef putData:data metadata:nil
completion:^(FIRStorageMetadata *metadata, NSError *error) {
    if (error != nil) {
        // Uh-oh, an error occurred!
    } else {
        // Metadata contains file metadata such as size, content-type, and download URL.
    }
}];
```



```
        NSURL downloadURL = metadata.downloadURL;
    }
}];
```

```
// Swift
let data: NSData! = ...
let uploadTask = riversRef.putData(data, metadata: nil) { metadata, error in
    if (error != nil) {
        // Uh-oh, an error occurred!
    } else {
        // Metadata contains file metadata such as size, content-type, and download URL.
        let downloadURL = metadata!.downloadURL
    }
}
```

Read Storage online: <https://riptutorial.com/firebase/topic/4281/storage>

Chapter 18: Structuring Data

Introduction

Firebase database is a NoSQL database that stores its data in the form of hierarchical JSON objects. There are no tables or records of any form as an SQL database would normally have, just nodes that make up a key-value structure.

Data Normalization

In order to have a properly designed database structure, the data requirements must be thoroughly outlined and forethought. The structure in this case should be normalized; the more flat the JSON tree, the faster data-access is.

Examples

Do's and Don'ts

The Wrong Way

Consider the following structure

```
{
  "users": {

    // Uniquely generated IDs for children is common practice,
    // it's actually really useful for automating child creation.
    // Auto-incrementing an integer for a key can be problematic when a child is removed.

    "-KH3Cx0KFvSQELIYZezv": {
      "name": "Jon Snow",
      "aboutMe": "I know nothing...",
      "posts": {
        "post1": {
          "body": "Different roads sometimes leads to the same castle",
          "isHidden": false
        },
        "post2": { ... },
        // Possibly more posts
      }
    },
    "-KH3Dx2KFdSLerIYZcgk": { ... },      // Another user
    // A lot more users here
  }
}
```

This is a great example of what **NOT** to do. Multi-nested structures such as the one above can be very problematic and could cause a huge performance setback.

The way Firebase accesses a node is by downloading all the children's data, then iterating over all same-level nodes (all parents' children). Now, imagine a database with several *users*, each having

hundreds (or even thousands) of *posts*. Accessing a *post* in this case could potentially load hundreds of megabytes of unused data. In a more complicated application, the nesting could be deeper than just 4 layers, which would result in more useless downloads and iterations.

The Right Way

Flattening the same structure would look like this

```
{
  // "users" should not contain any of the posts' data
  "users": {
    "-KH3Cx0KFvSQELIYZezv": {
      "name": "Jon Snow",
      "aboutMe": "I know nothing..."
    },
    "-KH3Dx2KFdSLerIYZcgk": { ... },
    // More users
  },

  // Posts can be accessed provided a user key
  "posts": {
    "-KH3Cx0KFvSQELIYZezv": {      // Jon Snow's posts
      "post1": {
        "body": "Different roads sometimes leads to the same castle",
        "isHidden": false
      },
      "post2": { ... },
      // Possibly more posts
    },
    "-KH3Dx2KFdSLerIYZcgk": { ... },
    // other users' posts
  }
}
```

This spares a huge amount of overhead by iterating over less nodes to access a target object. All *users* that do not have any posts would not exist in the *posts* branch, and so iterating over those users in *the wrong way* above is completely useless.

Two-Way Relationships

The following is an example of a simple and minimal college database that uses two-way relationships

```
{
  "students": {
    "-SL3Cs0KFvDMQLIYZEzv": {
      "name": "Godric Gryffindor",
      "id": "900130309",
      "courses": {
        "potions": true,
        "charms": true,
        "transfiguration": true,
      }
    },
    "-SL3ws2KvZQLTYMqzSas": {
      "name": "Salazar Slytherin",
```

```

    "id": "900132319",
    "courses": {
      "potions": true,
      "herbs": true,
      "muggleStudies": true,
    }
  },
  "-SL3ns2OtARSTUMywqWt": { ... },
  // More students here
},

"courses": {
  "potions": {
    "code": "CHEM305",
    "enrolledStudents": {
      "-SL3Cs0KFvDMQLIYZEzv": true,      // Godric Gryffindor
      "-SL3ws2KvZQLTYMqzSas": true,      // Salazar Slytherin
      // More students
    }
  },
  "muggleStuddies": {
    "code": "SOC215",
    "enrolledStudents": {
      "-SL3ws2KvZQLTYMqzSas": true,      // Salazar Slytherin
      "-SL3ns2OtARSTUMywqWt": true,      // Some other student
      // More students
    }
  },
  // More courses
}
}

```

Note that each student has a list of *courses* and each course has a list of enrolled *students*.

Redundancy is not always a bad approach. It's true that it costs storage space and having to deal with multiple entries' updating when deleting or editing a duplicated node; however, in some scenarios where data is not updated often, having two-way relationships could ease the fetching/writing process significantly.

In most scenarios where an SQL-like query seems needed, inverting the data and creating two-way relationships is usually the solution.

Consider an application using the database above that requires the ability to:

1. List the courses a certain student is taking **and**...
2. List all the students in a certain course

If the database structure had been one-directional, it would be incredibly slower to scan or query for one of the two requirements above. In some scenarios, redundancy makes frequent operations faster and much more efficient which, on the long run, makes the duplications' cost negligible.

Read Structuring Data online: <https://riptutorial.com/firebase/topic/8912/structuring-data>

Chapter 19: Using Firebase with Node

Examples

Hello World Firebase Realtime Database in Node

System Requirements:

- [Node JS](#)

Getting Started

1. First Go to [Firebase Console](#) and Create New Project.
2. After Creating the project, in project click on settings icon besides project Name in left sidebar and select Permissions.
3. On Permissions Page Click on Service accounts in left sidebar then click on Create Service Account
4. In the popup window enter your service account name and choose Account Role and select Furnish a new private key and after that select JSON and click Create(Leave Enable Google App Domain-wide Delegation Unchecked).
5. When you click create it will download a JSON file with your Account Credentials, just save the file Anywhere in your System.
6. Next step is to Create a Database in your Firebase Console for which Go to Firebase Console and click on Database in left-sidebar. After that just create a new Database Object with Name user_data with some dummy value.
7. Now your Firebase Database project is setup now simply copy following code in your project directory.

```
//Loading Firebase Package
var firebase = require("firebase");

/**
 * Update your Firebase Project
 * Credentials and Firebase Database
 * URL
 */
firebase.initializeApp({
  serviceAccount: "<path to Firebase Credentials Json File>",
  databaseURL: "<Firebase Database URL>"
}); //by adding your credentials, you get authorized to read and write from the database

/**
 * Loading Firebase Database and refering
 * to user_data Object from the Database
 */
var db = firebase.database();
var ref = db.ref("/user_data"); //Set the current directory you are working in

/**
 * Setting Data Object Value
```

```

*/
ref.set([
{
    id:20,
    name:"Jane Doe",
    email:"jane@doe.com",
    website:"https://jane.foo.bar"
},
{
    id:21,
    name:"John doe",
    email:"john@doe.com",
    website:"https://foo.bar"
}
]);

/**
 * Pushing New Value
 * in the Database Object
 */
ref.push({
    id:22,
    name:"Jane Doe",
    email:"jane@doe.com",
    website:"https://jane.foo.bar"
});

/**
 * Reading Value from
 * Firebase Data Object
 */
ref.once("value", function(snapshot) {
    var data = snapshot.val();    //Data is in JSON format.
    console.log(data);
});

```

8. Just change with the JSON Credentials file URL(For starters just copy the credentials file in Same folder and in index.js file just add the credentials File Name).

9. Next step is to change the in index.js with actual Firebase Database URL, you will be able to find this URL in Firebase Console in Database Tab, The URL will be like *<https://firebaseio.com/>*.

10. The final step is to do

```
npm install firebase
```

11. After Executing above command NPM will install necessary packages required for Firebase. Finally to run and test project execute

```
node index.js
```

What does the project actually do?

The project loads the Data from cloud based Firebase Database. The project also demonstrates how to Write and Read data from a Firebase Data Object.

In order to view your data get updated in realtime, go to [your console](#) click on the project you made, and on the left, hit Database. There, you can see your data get updated in real-time, along with their values.

Firestore-queue and worker

You can push tasks or data to the firestore realtime database and run a worker which listens to the firestore queue to run some background processes

Setup firestore

1. Create a Firestore project in the Firestore console, if you don't already have one. If you already have an existing Google project associated with your app, click Import Google Project. Otherwise, click Create New Project..
2. Click settings icon and select Permissions.
3. Select Service accounts from the menu on the left.
4. Click Create service account.

Enter a name for your service account.

You can optionally customize the ID from the one automatically generated from the name.

Choose Project > Editor from the Role dropdown.

Select Furnish a new private key and leave the Key type as JSON.

Leave Enable Google Apps Domain-wide Delegation unselected.

Click Create

5. When you create the service account, a JSON file containing your service account's credentials is downloaded for you. You'll need this to initialize the SDK in the server.

Setup server

Install firestore-queue using npm in your nodejs app

```
npm install firestore firestore-queue --save
```

Once you've installed firestore and firestore-queue, you can get started by creating a new Queue and passing it your Firestore reference and a processing function.

Now lets create a firestore queue task from the app when a new user is created and set worker to listen for firestore-queue task and send an email to the created users mail.

***server.js**

```
var app=express();
var Queue = require('firebase-queue'),
    Firebase = require('firebase');
```

Update your Firebase Project Credentials and Firebase Database URL

```
var firebase = Firebase.initializeApp({
  serviceAccount: "path/to/serviceAccountCredentials.json",
  databaseURL: "https://databaseName.firebaseio.com"
});
```

or you can input firebase credentials directly as below

```
var firebase = Firebase.initializeApp({
  serviceAccount: {
    projectId: "projectId",
    clientEmail: "foo@projectId.iam.gserviceaccount.com",
    privateKey: "-----BEGIN PRIVATE KEY-----\nkey\n-----END PRIVATE KEY-----\n"
  },
  databaseURL: "https://databaseName.firebaseio.com"
});

var refQueue = firebase.database().ref("queue/tasks");

createUser = function(email, password){
  var user = {
    username: email,
    password: password
  };
  user = new db.users(user);
  user.save(function(err, user){
    if(!err){
      refQueue.push({case: "NEW_USER", data: user});
    }
  })
}

createUser("abc@abc.com", "password");
```

*worker.js

```
var Queue = require('firebase-queue'),
    Firebase = require('firebase');

//Update your Firebase Project Credentials and Firebase Database URL by one of the way
specified in server.js
var firebase = Firebase.initializeApp({
  serviceAccount: "path/to/serviceAccountCredentials.json",
  databaseURL: "https://databaseName.firebaseio.com"
});

var refQueue = firebase.database().ref("queue");

var queue = new Queue(refQueue, function(data, progress, resolve, reject) {
  switch(data.case){
    case "NEW_USER":
```



```
        sendMail(data.data.email);
        console.log("user created");
        //sendMail function is not an inbuilt function and will not work unless you define
and implement the function
        break;

    // Finish the task asynchronously
    setTimeout(function() {
        resolve();
    }, 1000);
});
```

run server and worker separately and test around with firebase queue

```
node server.js

node worker.js
```

Read Using Firebase with Node online: <https://riptutorial.com/firebase/topic/6443/using-firebase-with-node>

Credits

S. No	Chapters	Contributors
1	Getting started with firebase	Ami Hollander , Community , Dan Levy , Devid Farinelli , ErstwhileIII , Gabriele Mariotti , RyanM , Sneh Pandya , TwiterZX , Vishal Vishwakarma
2	Cloud Functions for Firebase	Vishal Vishwakarma
3	Crash Reporting	Gabriele Mariotti
4	Database Rules	Frank van Puffelen , Gabriele Mariotti , riggaroo , Sasxa , Velko Ivanov
5	Email Verification after Sign Up	Rexford
6	Firebase Realtime Database with Android	Dhaval Solanki , Frank van Puffelen
7	Firebase Console	Priyank Bhojak
8	Firebase Offline Capabilities	Francisco Durdin Garcia
9	Firebase Queue	Vladimir Gabrielyan
10	FirebaseUI	Mike McDonald
11	FirebaseUI (Android)	Willie Chalmers III
12	How do I listen for errors when accessing the database?	Frank van Puffelen , ThunderStruct
13	How to get push key value from Firebase Database?	cutiko
14	How to use FirebaseRecyclerAdapter instead of RecyclerViewAdapter?	Dhaval Solanki
15	How to use the Firebase	Devid Farinelli , eikooc , Frank van Puffelen , Ron Royston

	Database to keep a list of Firebase Authentication users	
16	Push notification from custom server	Aawaz Gyawali
17	Storage	Mike McDonald
18	Structuring Data	ThunderStruct
19	Using Firebase with Node	Akshay Khale , Laurel , Noushad PP , Shiven